Milvus v1.1.0

Contents

About Milvus

What is Milvus

Overview

Milvus is an open-source vector database that is highly flexible, reliable, and blazing fast. It supports adding, deleting, updating, and near real-time search of vectors on a trillion-byte scale. A comprehensive set of intuitive APIs, and support for multiple widely adopted index libraries (e.g., Faiss, NMSLIB, and Annoy), simplifies the process of choosing the right index type for a given scenario. Additionally, support for scalar data filtering ensures Milvus maintains a high recall rate and remains adaptable.

Milvus runs on a client-server model. At a high-level, it operates as follows:

- The Milvus server includes the Milvus Core and Meta Store.
  - Milvus Core stores and manages vectors and scalar data.
  - Meta Store stores and manages metadata in SQLite for testing or MySQL for production.
- On the client side, Milvus provides SDKs in Python, Java, Go, and C++, as well as RESTful APIs.

Milvus was released under the open-source Apache License 2.0 in October 2019, and its source code was made available on GitHub. In June 2021, Milvus graduated from the LF AI & Data Foundation's incubator program.

The Milvus server runs on a standalone node. For scenarios involving large datasets or requiring high concurrency consider Mishards, our cluster sharding middleware.

Overall architecture

Scenarios

Milvus has been used in hundreds of organizations and institutions worldwide including the following scenarios:

- Image, video, and audio search.
- Recommender systems, chatbots, and other text search fields.
- New drug discovery, genetic screening, and other biomedical fields.

See Scenarios for more information.

Key features

Heterogeneous computing

- Optimizes search and indexing performance on GPU.
- Searches trillion-byte scale datasets in milliseconds.
- Manages inserting, deleting, updating, and querying vector data in a dynamic environment.

Compatible with mainstream libraries, metrics, and tooling

- Offers support for Faiss, NMSLIB, and Annoy libraries.
- Supports graph- and tree-based indexes as well as quantization.
- Measures similarity using Euclidean distance (L2), inner product, Hamming distance, Jaccard distance, and more.
- Monitors and visualizes runtime metrics using Prometheus and Grafana.

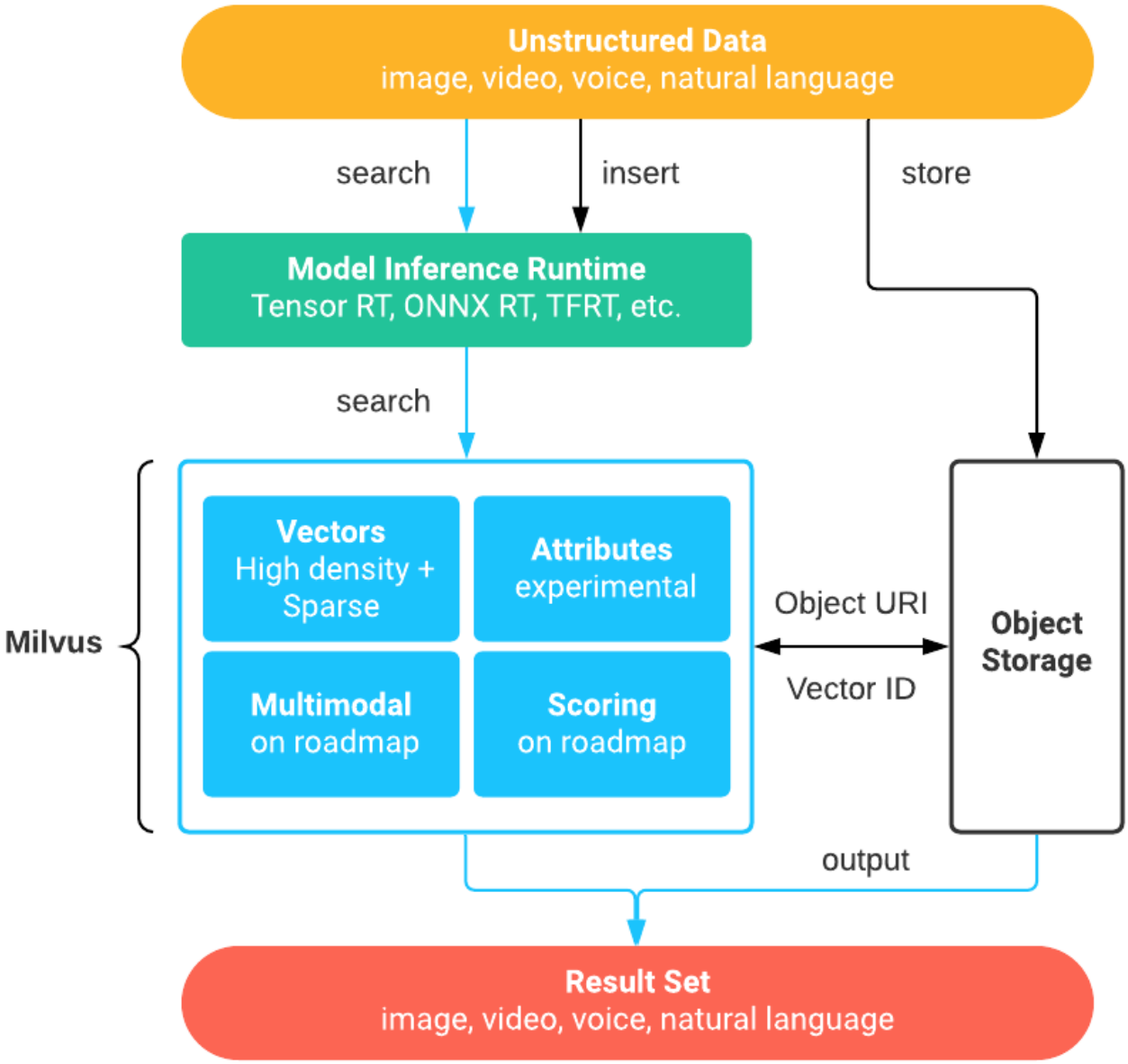

Figure 1: Milvus architecture

Near-real-time (NRT) search

- Newly inserted datasets are available for search in one second or less.

Scalar field filtering (coming soon)

- Makes search more flexible by allowing data to be filtered more granularly.

Milvus distributions

Milvus is available in CPU-only and GPU-enabled distributions:

The CPU-only Milvus distribution relies on CPU exclusively to search and build indexes.

The GPU-enabled Milvus distribution supports GPU acceleration for search and index building. For example, CPU can be used for search while GPU is used for index building, improving query efficiency.

For GPUs that support CUDA, the GPU-enabled Milvus distribution can be used to achieve much better search performance when working with large-scale datasets.

See Milvus Distributions for more information..

Join our community

Before joining our developer community, please take some time to read our code contribution guidelines.

For questions about Milvus' functionality or SDKs, join our GitHub Discussions or Slack channel.

Milvus Distributions

Overview

Milvus is available in CPU-only and GPU-enabled distributions:

The CPU-only Milvus distribution relies on CPU exclusively to search and build indexes.

The GPU-enabled Milvus distribution supports GPU acceleration for search and index building. For example, CPU can be used for search while GPU is used for index building, improving query efficiency.

For GPUs that support CUDA, the GPU-enabled Milvus distribution can be used to achieve much better search performance when working with large-scale datasets.

CPU-only MilvusGPU-enabled Milvus

Indexes for CPU-only Milvus

Milvus maps different embedding types with different index types. Click the tab below to view the index types supporting your embedding type.

Floating point embeddings Binary embeddings

Index type	Indexing with CPU	Indexing with GPU	Search with CPU	Search with GPU
FLAT	N/A	N/A	✔	✖
IVF_FLAT	✔	✖	✔	✖
IVF_SQ8	✔	✖	✔	✖
IVF_PQ	✔	✖	✔	✖
RNSG	✔	✖	✔	✖
HNSW	✔	✖	✔	✖
Annoy	✔	✖	✔	✖

Index type	Indexing with CPU	Indexing with GPU	Search with CPU	Search with GPU
FLAT	N/A	N/A	✔	×
IVF_FLAT	✔	×	✔	×

Announcement Letter

Milvus 1.0: The World's Most Popular Open-Source Vector Database Just Got Better

Zilliz is proud to announce the release of Milvus v1.0. After months of extensive testing Milvus v1.0, which is based on a stable version of Milvus v0.10.6, is available for use.

Milvus v1.0 offers the following key features:

- Support for mainstream similarity metrics, including Euclidean distance, inner product, Hamming distance, Jaccard coefficient, and more.
- Integration with, and improvements to, SOTA ANNs algorithms, including Faiss, Hnswlib, Annoy, NSG, and more.
- Scale-out capability through the Mishards sharding proxy.
- Support for processors commonly used in AI scenarios, including X86, Nvidia GPU, Xilinx FPGA, and more.

See the Release Notes for additional Milvus v1.0 features.

Milvus is an ongoing open-source software (OSS) project. Its first major release has the following implications for users:

- Milvus v1.0 will receive long-term support (3+ years).
- The most stable Milvus release to-date is well structured and ready for integration with existing AI ecosystems.

The first version of Milvus with long-term support

Thanks in part to sponsorship from Zilliz, the Milvus community will provide bug fix support for Milvus v1.0 until December 31st, 2024. New features will be available only in releases following v1.0.

See The Milvus release guideline for information about release cadences and more.

Toolchain enhancements and seamless AI ecosystem integration

Beginning with v1.0, Milvus' toolchain will be a primary development focus. We plan to create the necessary tooling and utilities to meet the needs of the Milvus user community.

Stability makes integrating Milvus with AI ecosystems a breeze. We are seeking further collaboration between the Milvus community and other AI-focused OSS communities. We encourage contributions to the new AI ASICs (application-specific integrated circuits) in Milvus.

The future of Milvus

We believe Milvus has a bright future thanks to the following factors:

- Regular contributions from developers in the Milvus community.
- Support for integration with any cloud-native environment.

We have drafted community charters to help guide, nurture, and advance the Milvus community as our technology and user base grows. The charters include several technical decisions made to attract more participants to the community.

- Golang will now be used to develop the Milvus engine however, the ANNS algorithm component will still be developed in C++.
- The forthcoming distributed version of Milvus will use existing cloud components as much as possible.

We are thrilled to partner with the open-source software community to build the next-generation cloud data fabric made for AI. Let's get to work!

Don’t be a stranger

- Find or contribute to Milvus on GitHub
- Interact with the community via Slack.
- Connect with us on Twitter. ## Milvus Data Migration Guide

This migration guide deals with migrating data from Milvus v0.7.0~0.10.6 to Milvus v1.0.0.

Milvus v0.11.0 and versions earlier than v0.7.0 are incompatible with v1.0.0.

MilvusDM supports migrating data from Milvus v0.10.x to v1.0.0

Step 1: Stop Current Version of Milvus

Stop the current version of Milvus:

```
docker stop [Your_milvus_container_id]
```

Delete /conf, /logs, and /wal under /milvus:

```
cd ~/milvus
sudo rm -rf ./conf
sudo rm -rf ./logs
sudo rm -rf ./wal
```

Save a copy of the logs folder if you want to retain log files.

Step 2: Download the v1.0.0 Configuration File

Create a conf directory and download the v1.0.0 configuration file:

```
mkdir conf
cd conf
wget https://raw.githubusercontent.com/milvus-io/milvus/v1.1.0/core/conf/demo/server_config.yaml
```

If the download is unsuccessful, you can open the download URL on a web page, create a new file with the same name in the conf directory, then save the web page content by copying it to the new file.

Step 3: Update the Server Address of MySQL/SQLite

```
vim ./server_config.yaml
```

Ensure that the MySQL/SQLite server address specified in `general.meta_uri` matches the server address specified in `db_config.backend_url`. If you use MySQL to manage metadata, the configuration information appears as follows:

```
general:
timezone:UTC+8
meta_uri: mysql://root:123456@<MySQL_server_host IP>:3306/milvus
```

Step 4: Download and Start Milvus v1.0.0

Download and run a Milvus v1.0.0 docker image using the same mapping path setting:

```
$ sudo docker run -d --name milvus_cpu_1.1.0 \
-p 19530:19530 \
-p 19121:19121 \
-v /home/$USER/milvus/db:/var/lib/milvus/db \
-v /home/$USER/milvus/conf:/var/lib/milvus/conf \
-v /home/$USER/milvus/logs:/var/lib/milvus/logs \
-v /home/$USER/milvus/wal:/var/lib/milvus/wal \
milvusdb/milvus:1.1.0-cpu-d050721-5e559c
```

Step 5: Install the Python SDK Corresponding to Milvus v1.0.0

```
pip3 install pymilvus==1.1.0
```

Step 6: Verify Data Correctness

Write and run a Python script to verify if the data is correct. # Concepts
Embedding Vector

Definition

An embedding vector is a series of numbers and can be considered as a matrix with only one row but multiple columns, such as [2,0,1,9,0,6,3,0].

An embedding vector includes information representing the characteristics of an object, such as RGB (red-green-blue) color descriptions. A color can be described by the proportions of red, green, and blue. An embedding vector in RGB could be [R, G, B].

Advantages

Advances in modern computer and machine learning technologies have led to massive amounts of multimedia data in diverse application fields such as real estate, pharmaceutical, and financial information services. A multimedia object cannot be simply described by alphanumeric data because a multimedia object have multiple dimensions of properties.

Instead, embedding vectors describe an object in a multi-dimensional, easily analyzable way, and are suitable to represent numeric or symbolic characteristics of multimedia content.

Embedding vectors are important for many different fields of machine learning and pattern recognition. Machine learning algorithms typically require a numerical representation of objects in order for the algorithms to perform statistical analysis.

Scenarios

Embedding vectors, with its effectiveness and practicality of numerically representing objects, are used widely in different fields of machine learning.

- Image processing

  Features can be gradient magnitudes, colors, grayscale intensities, edges, areas, and more. Embedding vectors are particularly popular in image processing because it is easy to define numeric attributes for images.

- Speech recognition

  Features can be sound lengths, noise levels, noise ratios, and more.

- Spam filtering

  Features can be IP addresses, text structures, frequencies of certain words, certain email headers, and more.

Vector Index

Vector index

Vector index is a time-efficient and space-efficient data structure built on vectors through a certain mathematical model. Through the vector index, we can efficiently query several vectors similar to the target vector.

Since accurate retrieval is usually very time-consuming, most of the vector index types of Milvus use ANNS (Approximate Nearest Neighbors Search). Compared with accurate retrieval, the core idea of ANNS is no longer limited to returning the most accurate result, but only searching for neighbors of the target. ANNS improves retrieval efficiency by sacrificing accuracy within an acceptable range.

According to the implementation methods, the ANNS vector index can be divided into four categories:

- Tree-based index
- Graph-based index
- Hash-based index
- Quantization-based index

The following table classifies the indexes that Milvus supports:

Supported index

Classification

Scenario

FLAT

N/A

Has a relatively small dataset.

Requires a 100% recall rate.

IVF_FLAT

Quantization-based index

High-speed query.

Requires a recall rate as high as possible.

IVF_SQ8

Quantization-based index

High-speed query.

Limited disk and memory capacity.

Has CPU resources only.

IVF_SQ8H

Quantization-based index

High-speed query.

Limited disk, memory, and graphics memory capacities.

IVF_PQ

Quantization-based index

RNSG

Graph-based index

HNSW

Graph-based index

Annoy

Tree-based index

Vector field and index

To improve query performance, you can specify an index type for each vector field. Currently, a vector field only supports one index type, Milvus will automatically delete the old index when switching the index type.

Create indexes

When the `create_index()` method is called, Milvus synchronously indexes the existing data on this field. Whenever the size of the inserted data reaches the `index_file_size`, Milvus automatically creates an index for it in the background.

When the inserted data segment is less than 4096 rows, Milvus does not index it.

Index by segment

Milvus stores massive data in sections. When indexing, Milvus creates an index for each data segment separately.

Build indexes during free time

It is known that indexing is a resource-consuming and time-consuming task. When the query task and indexing task are concurrent, Milvus preferentially allocates computing resources to the query task, that is, any query command will interrupt the indexing task being executed in the background. After that, only when the user does not send the query task for 5 seconds, Milvus resumes the indexing task in the background. Besides, if the data segment specified by the query command has not been built into the specified index, Milvus will do an exhaustive search directly within the segment.

Supported vector indexes

FLAT

If FLAT index is used, the vectors are stored in an array of float/binary data without any compression. during searching vectors, all indexed vectors are decoded sequentially and compared to the query vectors.

FLAT index provides 100% query recall rate. Compared to other indexes, it is the most efficient indexing method when the number of queries is small.

- Search parameters

Parameter	Description	Range
`metric_type`	[Optional] The chosen distance metric.	See Supported Metrics.

IVF_FLAT

IVF (Inverted File) is an index type based on quantization. It divides the points in space into `nlist` units by clustering method. during searching vectors, it compares the distances between the target vector and the center of all the units, and then select the `nprobe` nearest unit. Then, it compares all the vectors in these selected cells to get the final result.

IVF_FLAT is the most basic IVF index, and the encoded data stored in each unit is consistent with the original data.

- Index building parameters

Parameter	Description	Range
`nlist`	Number of cluster units	[1, 65536]

- Search parameters

Parameter	Description	Range
`nprobe`	Number of units to query	CPU: [1, nlist] GPU: [1, min(2048, nlist)]

IVF_SQ8

IVF_SQ8 does scalar quantization for each vector placed in the unit based on IVF. Scalar quantization converts each dimension of the original vector from a 4-byte floating-point number to a 1-byte unsigned integer, so the IVF_SQ8 index file occupies much less space than the IVF_FLAT index file. However, scalar quantization results in a loss of accuracy during searching vectors.

- IVF_SQ8 has the same index building parameters as IVF_FLAT.
- IVF_SQ8 has the same search parameters as IVF_FLAT.

IVF_SQ8H

Optimized version of IVF_SQ8 that requires both CPU and GPU to work. Unlike IVF_SQ8, IVF_SQ8H uses a GPU-based coarse quantizer, which greatly reduces time to quantize.

IVF_SQ8H is an IVF_SQ8 index that optimizes query execution.

The query method is as follows:

- If `nq` $\geq$ `gpu_search_threshold`, GPU handles the entire query task.
- If `nq` $<$ `gpu_search_threshold`, GPU handles the task of retrieving the `nprobe` nearest unit in the IVF index file, and CPU handles the rest.
- IVF_SQ8H has the same index building parameters as IVF_FLAT.
- IVF_SQ8H has the same search parameters as IVF_FLAT.

IVF_PQ

`PQ` (Product Quantization) uniformly decomposes the original high-dimensional vector space into Cartesian products of `m` low-dimensional vector spaces, and then quantizes the decomposed low-dimensional vector spaces. In the end, each vector is stored in `m` $\times$ `nbits` bits. Instead of calculating the distances between the target vector and the center of all the units, product quantization enables the calculation of distances between the target vector and the clustering center of each low-dimensional space and greatly reduces the time complexity and space complexity of the algorithm.

IVF_PQ quantizes the products of vectors before IVF index clustering. Its index file is even smaller than IVF_SQ8, but it also causes a loss of accuracy during searching vectors.

Index building parameters and search parameters vary with Milvus distribution. Select your Milvus distribution first.

CPU-only Milvus GPU-enabled Milvus

- Index building parameters

Parameter	Description	Range
`nlist`	Number of cluster units	[1, 65536]
`m`	Number of factors of product quantization	dim $\equiv$ 0 (mod m)
`nbits`	[Optional] Number of bits in which each low-dimensional vector is stored.	[1, 16] (8 by default)

- Search parameters

Parameter	Description	Range
`nprobe`	Number of units to query	[1, nlist]

- Index building parameters

Parameter	Description	Range
nlist	Number of cluster units	[1, 65536]
m	Number of factors of product quantization	m ∈ {1, 2, 3, 4, 8, 12, 16, 20, 24, 28, 32, 40, 48, 56, 64, 96}, and (dim / m) ∈ {1, 2, 3, 4, 6, 8, 10, 12, 16, 20, 24, 28, 32}.(m x 1024) ≤ MaxSharedMemPerBlock of your graphics card.
nbits	[Optional] Number of bits in which each low-dimensional vector is stored.	8

If the value of m does not fall into the specified range for GPU indexing but into the range of CPU indexing, Milvus switches to using CPU to build the index (click the button above to view the range supported by CPU-enabled Milvus).

If the specified value of nbits is between 1 and 16 but not 8, the system switches back to CPU-only Milvus.

- Search parameters

Parameter	Description	Range
nprobe	Number of units to query	[1, min(2048, nlist)]

If the value of nprobe does not fall into the specified range but into the range for CPU search, Milvus switches to CPU search (click the button above to view the range supported by CPU-enabled Milvus).

RNSG

RNSG (Refined Navigating Spreading-out Graph) is a graph-based indexing algorithm. It sets the center position of the whole image as a navigation point, and then uses a specific edge selection strategy to control the out-degree of each point (less than or equal to out_degree). Therefore, it can reduce memory usage and quickly locate the target position nearby during searching vectors.

The graph construction process of RNSG is as follows:

1. Find knng nearest neighbors for each point.
2. Iterate at least search_length times based on knng nearest neighbor nodes to select candidate_pool_size possible nearest neighbor nodes.
3. Construct the out-edge of each point in the selected candidate_pool_size nodes according to the edge selection strategy.

The query process is similar to the graph building process. It starts from the navigation point and iterate at least search_length times to get the final result.

- Index building parameters

Parameter	Description	Range
out_degree	Maximum out-degree of the node	[5, 300]
candidate_pool_size	Candidate pool size of the node	[50, 1000]
search_length	Number of query iterations	[10, 300]
knng	Number of nearest neighbors	[5, 300]

- Search parameters

Parameter	Description	Range
search_length	Number of query iterations	[10, 300]

HNSW

HNSW (Hierarchical Small World Graph) is a graph-based indexing algorithm. It builds a multi-layer navigation structure for an image according to certain rules. In this structure, the upper layers are more sparse and the distances between nodes are farther; the lower layers are denser and the distances between nodes are closer. The search starts from the uppermost layer, finds the node closest to the target in this layer, and then enters the next layer to begin another search. After multiple iterations, it can quickly approach the target position.

In order to improve performance, HNSW limits the maximum degree of nodes on each layer of the graph to M. In addition, you can use efConstruction (when building index) or ef (when searching targets) to specify a search range.

- Index building parameters

Parameter	Description	Range
M	Maximum degree of the node	[4, 64]
efConstruction	Search scope	[8, 512]

- Search parameters

Parameter	Description	Range
ef	Search scope	[top_k, 32768]

Annoy

Annoy (Approximate Nearest Neighbors Oh Yeah) is an index that uses a hyperplane to divide a high-dimensional space into multiple subspaces, and then stores them in a tree structure.

When searching for vectors, Annoy follows the tree structure to find subspaces closer to the target vector, and then compares all the vectors in these subspaces (The number of vectors being compared should not be less than search_k) to obtain the final result. Obviously, when the target vector is close to the edge of a certain subspace, sometimes it is necessary to greatly increase the number of searched subspaces to obtain a high recall rate. Therefore, Annoy uses n_trees different methods to divide the whole space, and searches all the dividing methods simultaneously to reduce the probability that the target vector is always at the edge of the subspace.

- Index building parameters

Parameter	Description	Range
n_trees	The number of methods of space division.	[1, 1024]

- Search parameters

Parameter	Description	Range
search_k	The number of nodes to search. -1 means 5% of the whole data.	{-1} ∪ [top_k, n × n_trees]

How to choose an index

To learn how to choose an appropriate index for your application scenarios, please read How to Select an Index in Milvus.

To learn how to choose an appropriate index for a metric, see Distance Metrics.

FAQ

Does IVF_SQ8 differ from IVF_SQ8H in terms of recall rate?

No, they have the same recall rate for the same dataset.

What is the difference between FLAT index and IVF_FLAT index?

IVF_FLAT index divides a vector space into nlist clusters. If you keep the default value of nlist as 16384, Milvus compares the distances between the target vector and the centers of all 16384 clusters to get nprobe nearest clusters. Then Milvus compares the distances between the target vector and the vectors in the selected clusters to get the nearest vectors. Unlike IVF_FLAT, FLAT directly compares the distances between the target vector and each and every vector.

Therefore, when the total number of vectors approximately equals nlist, IVF_FLAT and FLAT has little difference in the way of calculation required and search performance. But as the number of vectors grows to two times, three times, or n times of nlist, IVF_FLAT index begins to show increasingly greater advantages.

See How to Choose an Index in Milvus for more information.

Bibliography

- RNSG: Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph
- HNSW: Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs
- Annoy: Nearest neighbors and vector models part 2 algorithms and data structures

Storage Concepts

Partition and segment

When creating a collection, Milvus controls the size of a data segment according to the `index_file_size`. Also, Milvus provides partition function, you can divide the data into multiple partitions as needed. Reasonable organization and division of data can effectively improve query performance.

Segment

To process massive data, Milvus segments the data and each segment of data has tens or even hundreds of thousands of entities. Milvus separates the data in each segment by fields and stores the data in each field a data file. In the current version, an entity contains only one ID field and one vector field, so each segmented data file mainly includes a UID file and an original vector data file.

The size of a segment is determined by the `index_file_size` (1,024 MB by default and 4,096 MB at most) when Milvus is creating the collection.

When building indexes, Milvus builds an index for each segment in the collection in order and stores the index into a separate file. Index files are independent of each other. Indexing can significantly improve retrieval performance.

Partition

After a collection has accumulated massive data, the query performance gradually declines. In some scenarios, only part of the data in a collection needs to be queried, so Milvus divides the data in the collection into multiple parts on physical storage based on certain rules. Such operation is called partitioning. Each partition can contain multiple segments.

A partition is identified by a tag. When inserting vector data, you can use the tag to specify which partition to insert the data into. When querying vector data, you can use the tag to specify the partition where the query should be executed. Milvus supports both the exact matching and regular expression matching for partition tags.

Each collection can have 4096 partitions at most.

The relationship between collections, partitions and segments

The relationship between collections, partitions, and segments is as follows:

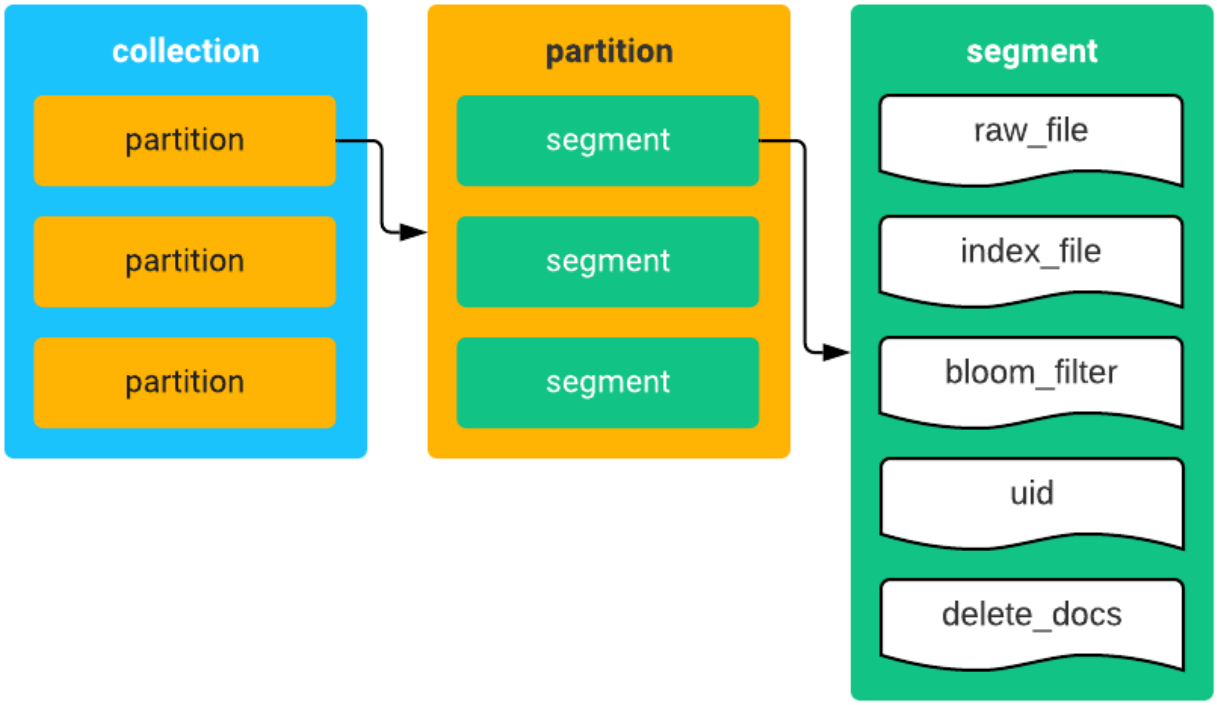

Figure 2: file

Each collection has a _default section. If no partition is specified when inserting data, Milvus inserts the data into the _default partition.

Metadata

Both the partition or segment are organizational forms of data in physical storage. When querying data, Milvus must know the location and status information of each data file on the physical storage, including the collection it belongs to, the number of entities it contains, the file size, the globally unique identifier, and the creation date, which are called metadata. In addition, the metadata also contains collection and partition information, including collection name, collection dimension, index type, partition label, and so on.

When data changes, the metadata should change accordingly and be easy to obtain. Therefore, it is an ideal choice to use a transactional database to manage metadata. Milvus provides SQLite or MySQL as a backend service for metadata and MySQL is recommended for production environments or distributed services.

The metadata back-end service is not responsible for storing entity data and indexes.

FAQ

Can I use SQL Server or PostgreSQL to store metadata in Milvus?

No, we only support storing metadata using SQLite or MySQL.

Storage Operations

Please read Storage Concepts before reading this article.

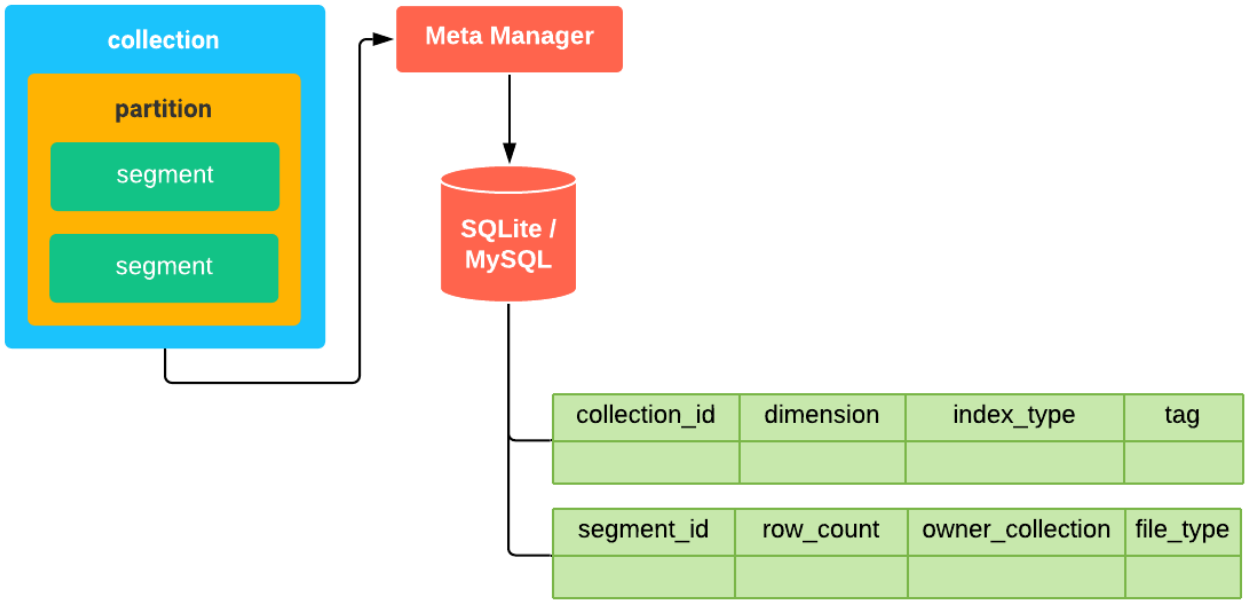

Figure 3: meta

Insert data

The client inserts data by calling the `insert` API, and the amount of inserted data cannot exceed 256 MB at a time. The process of data insertion is as follows:

1. After receiving the insert request, the server writes the data to the write ahead log (WAL).
2. After the request is successfully recorded to the log file, the server handles the insert operation.
3. The server writes data to a mutable buffer.

Each collection has an independent mutable buffer. The maximum capacity of each mutable buffer is 128 MB. The upper limit of the total mutable buffer capacity of all collections is determined by insert_buffer_size (by default 1 GB).

Flush data

There are three triggering mechanisms for data flushing in the buffer:

Timed trigger

The system triggers the data flushing task regularly. The interval is determined by the `auto_flush_interval` (by default 1 second).

The process of data flushing is as follows:

1. The system opens up a new mutable buffer area to accommodate the data to be inserted.
2. The system sets the previous mutable buffer as read-only (immutable buffer).
3. The system writes the data in the immutable buffer to the disk and writes the description information of the new data segment to the metadata backend service.

After completing the above process, the system has successfully created a segment.

Client trigger

The client calls the `flush` API to trigger the data flushing task. ### Trigger when the buffer reaches the upper limit

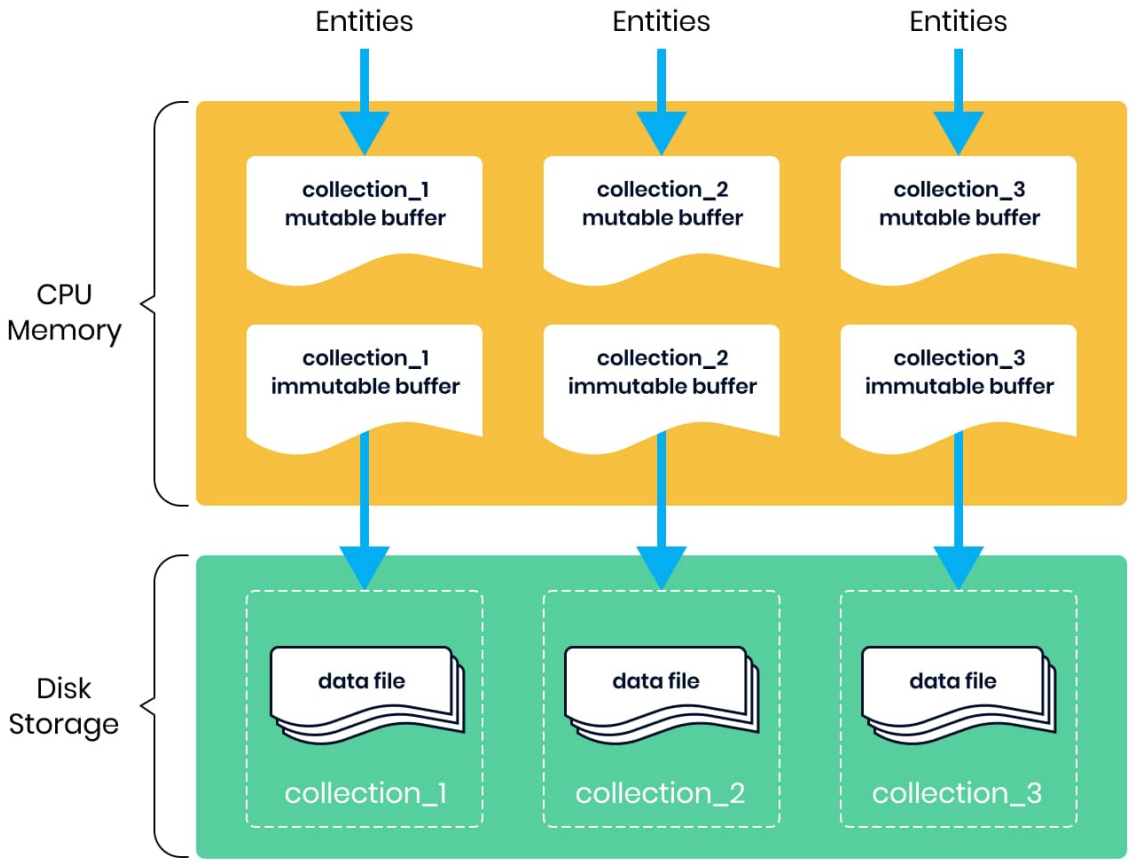

Figure 4: insert

When the accumulated data reaches the upper limit of the mutable buffer (128MB), the data flushing task is triggered.

All relevant files of each segment are stored in a folder named by the segment ID, such as a UID file that records the entity ID, a delete_docs file used to mark deleted entities, and a bloom-filter file used for quick entity search.

Please see the diagram in Partition and Segment for data files within the segment.

Merge data

Too many small data segments cause poor query performance. To address this problem, Milvus triggers the segment merge task in the background when needed. In other words, Milvus merges small data segments into new data segments, deletes the small data segments, and updates the metadata. The size of new data segments should not be less than the `index_file_size`.

The timings to trigger the merge task are as follows:

- When starting the service
- After completing a data flushing task
- Before building indexes
- After deleting the indexes

The indexed segments do not participate in the merge task.

Build indexes

Before building indexes, Milvus performs query operations on collections by brute-force search. To improve query performance, you can build a suitable index for the collection. After the index is built, Milvus generates an index file for each segment and simultaneously updates the metadata.

See Index Types for more information about building indexes.

Delete

Delete collections

1. The client calls the `drop_collection` API to delete a collection.
2. After receiving the request, the server marks the collection (including its partitions and segments) as deleted in the metadata. No new operations (such as insertion and query) can be performed on these collections.
3. The background cleanup task deletes the collection (including its partitions and segments) marked as deleted from the metadata and deletes the corresponding data files and folders from the disk. If there is an operation being performed on the collection before the delete operation is called, the segment is deleted after the previous operation is completed.

Delete partitions

1. The client calls the `drop_partition` API to delete a partition.
2. After receiving the request, the server marks the partition (including its segment) as deleted in the metadata.
3. The background cleanup task performs the same process as described in Delete collections to delete the partition and metadata.

Delete entities

Milvus created a delete_docs file for each segment to record the position of the vectors to be deleted within the segment.

Milvus uses a bloom filter to quickly determine whether an entity ID exists in a segment. Therefore, a file named bloom_filter is created under each segment.

The process of deleting an entity is as follows:

1. The client calls the `delete_entity_by_id` API to delete some entities in the collection.
2. After receiving the request, the server performs the following operations to delete the entities:

- If an entity is the insertion buffer, the server deletes the entity directly.
- Otherwise, based on the bloom filter of each segment, the server determines which segment contains the entity, and then updates the delete_docs and bloom_filter files of the segment.

Compact segments

When querying a segment, Milvus reads the entity data of the segment and the delete_docs file into memory. Although the deleted entities do not participate in the calculation, they are read into memory. Therefore, the more deleted entities in a segment, the more memory resources and disk space are wasted. To reduce such unnecessary resource consumption, Milvus provides data segment compaction operation, the process is as follows:

1. The client calls the `compact` API.
2. After receiving the request, the server writes the undeleted entities in the segment to a new segment based on the information recorded in delete_docs and marks the old segment as deleted. Afterward, the background cleanup task is responsible for cleaning the segments marked as deleted. If the old segments have been indexed, the indexes are rebuilt after the new segments are generated.

The compact operation ignores the segment where the deleted vector accounts for less than 10% of the entire data.

Read data

1. The client calls the `get_entity_by_id` API to read the original entity data.
2. After receiving the request, the server finds the segment where the entity is located through the bloom filter and returns the data corresponding to the entity ID.

Floating-point vectors are stored in Milvus as single-precision (float) data.

FAQ

Can I increase my storage by adding interfaces such as S3 or GlusterFS?

No, you cannot. Milvus does not support this feature for now.

Can I export data from Milvus?

Milvus DM supports exporting data from Milvus to HDF5 file.

MilvusDM is supported on Milvus 1.x only.

Similarity Metrics

In Milvus, distance metrics are used to measure similarities among vectors. Choosing a good distance metric helps improve the classification and clustering performance significantly.

The following table shows how these widely used distance metrics fit with various input data forms and Milvus indexes.

Floating point embeddings Binary embeddings

Distance Metrics

Index Types

Euclidean distance (L2)

FLAT

IVF_FLAT

IVF_SQ8

IVF_SQ8H

IVF_PQ

RNSG

HNSW

Annoy

Inner product (IP)

Distance Metrics

Index Types

Jaccard

Tanimoto

Hamming

FLAT

IVF_FLAT

Superstructure

Substructure

FLAT

Euclidean distance (L2)

Essentially, Euclidean distance measures the length of a segment that connects 2 points.

The formula for Euclidean distance is as follows:

$$d(\boldsymbol{a}, \boldsymbol{b}) = d(\boldsymbol{b}, \boldsymbol{a}) = \sqrt{\sum_{i=1}^{n} (b_i - a_i)^2}$$

Figure 5: euclidean

where a = (a1, a2,⋯, an) and b = (b1, b2,⋯, bn) are two points in n-dimensional Euclidean space

It’ s the most commonly used distance metric, and is very useful when the data is continuous.

Inner product (IP)

The IP distance between two embeddings are defined as follows:

$$p(A, B) = A \cdot B = \sum_{i=1}^{n} a_i \times b_i$$

Figure 6: ip

where A and B are embeddings, `||A||` and `||B||` are the norms of A and B.

IP is more useful if you are more interested in measuring the orientation but not the magnitude of the vectors.

If you use IP to calculate embeddings similarities, you must normalize your embeddings. After normalization, inner product equals cosine similarity.

Suppose X' is normalized from embedding X:

$$X' = (x'_1, x'_2, \ldots, x'_n), X' \in^n$$

Figure 7: normalize

The correlation between the two embeddings is as follows:

$$x'_i = \frac{x_i}{\| X \|} = \frac{x_i}{\sqrt{\sum_{i=1}^{n} (x_i)^2}}$$

Figure 8: normalization

Jaccard distance

Jaccard similarity coefficient measures the similarity between two sample sets, and is defined as the cardinality of the intersection of the defined sets divided by the cardinality of the union of them. It can only be applied to finite sample sets.

$$J(A,B) = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Figure 9: Jaccard similarity coefficient

Jaccard distance measures the dissimilarity between data sets, and is obtained by subtracting the Jaccard similarity coefficient from 1. For binary variables, Jaccard distance is equivalent to Tanimoto coefficient.

Tanimoto distance

For binary variables, the Tanimoto coefficient is equivalent to Jaccard distance:

In Milvus, the Tanimoto coefficient is only applicable for a binary variable, and for binary variables the Tanimoto coefficient ranges from 0 to +1 (where +1 is the highest similarity).

For binary variables, the formula of Tanimoto distance is:

The value ranges from 0 to +infinity.

$$d_j(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

Figure 10: Jaccard distance

$$T(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

Figure 11: tanimoto coefficient

Hamming distance

Hamming distance measures binary data strings. The distance between two strings of equal length is the number of bit positions at which the bits are different.

For example, suppose there are two strings 1101 1001 and 1001 1101.

11011001 ⊕ 10011101 = 01000100. Since, this contains two 1s, the Hamming distance, d (11011001, 10011101) = 2.

Superstructure

Superstructure is used to measure the similarity of a chemical structure and its superstructure. The less the value, the more similar the structure is to its superstructure. Only the vectors whose distance equals to 0 can be found now.

Superstructure similarity can be measured by:

Where

- B is the superstructure of A
- NA specifies the number of bits in the fingerprint of molecular A.
- NB specifies the number of bits in the fingerprint of molecular B.
- NAB specifies the number of shared bits in the fingerprint of molecular A and B.

Substructure

Substructure is used to measure the similarity of a chemical structure and its substructure. The less the value, the more similar the structure is to its substructure. Only the vectors whose distance equals to 0 can be found now.

Substructure similarity can be measured by:

Where

$$d_t = \frac{A \cdot B}{|A|^2 + |B|^2 - A \cdot B}$$

Figure 12: tanimoto distance

$$1 - \frac{N_{A\&B}}{N_A}$$

Figure 13: superstructure

$$1 - \frac{N_{A\&B}}{N_B}$$

Figure 14: substructure

- B is the substructure of A
- NA specifies the number of bits in the fingerprint of molecular A.
- NB specifies the number of bits in the fingerprint of molecular B.
- NAB specifies the number of shared bits in the fingerprint of molecular A and B.

FAQ

Why is the top1 result of a vector search not the search vector itself, if the metric type is inner product?

This occurs if you have not normalized the vectors when using inner product as the distance metric.

What is normalization? Why is normalization needed?

Normalization refers to the process of converting an embedding (vector) so that its norm equals 1. If you use Inner Product to calculate embeddings similarities, you must normalize your embeddings. After normalization, inner product equals cosine similarity.

See Wikipedia for more information.

Why do I get different results using Euclidean distance (L2) and inner product (IP) as the distance metric?

Check if the vectors are normalized. If not, you need to normalize the vectors first. Theoretically speaking, similarities worked out by L2 are different from similarities worked out by IP, if the vectors are not normalized.

Write Ahead Log

Write ahead log records insertion and deletion requests into the log file, and then the background thread writes it to the system. Once the requests are successfully written to the log file, the server returns success. This function enhances data reliability and reduces client blocking.

Data reliability

Write ahead log guarantees the atomicity of modification requests. All requests that receives success messages are completely written to the system. For requests that do not receive and respondence due to an unexpected system exit or an unexpected link disconnection, the operation is either succeed or fail. Whether the operation is successful can be confirmed by calling other interfaces. In addition, when the system restarts, some requests in the log file are re-executed if they have not been applied to the system state.

Buffer settings

The buffer size of the write ahead log is determined by the `wal.buffer_size`. To ensure the write performance of the write ahead log, we recommend setting the buffer size to at least twice the size of the data imported in a single batch.

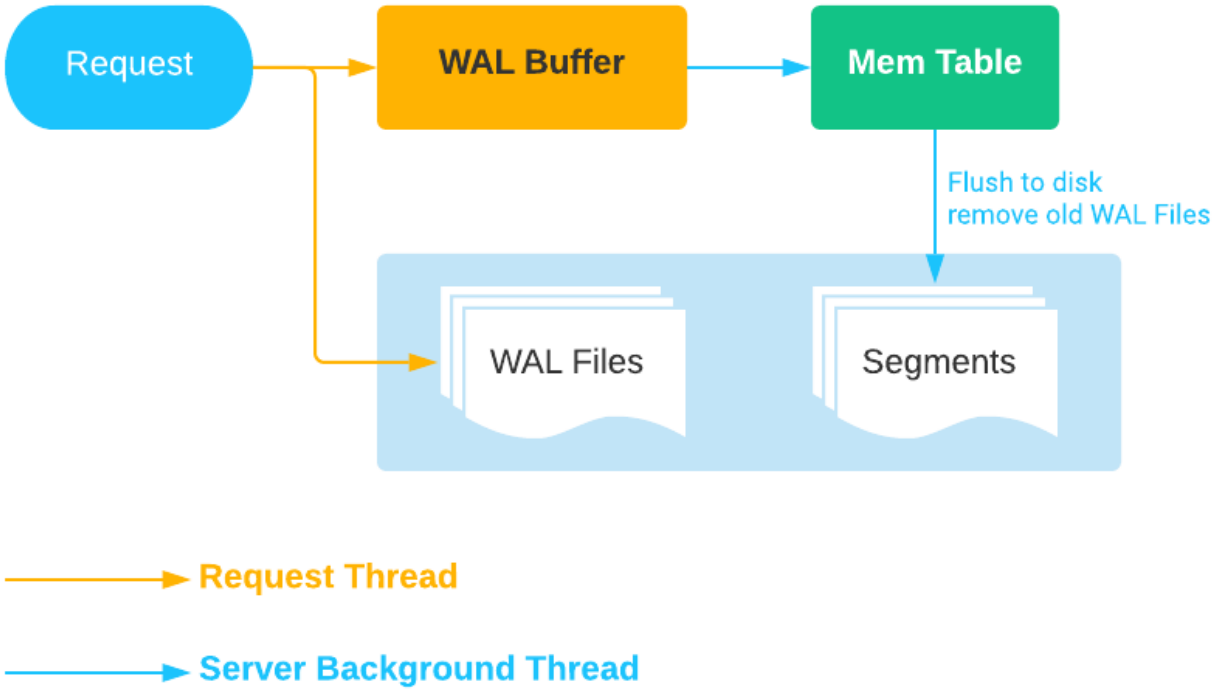

Figure 15: wal_structure

For more information about how to set wal.buffer_size, see Milvus configuration.

Delete old log files

Milvus automatically deletes log files that have been applied to the system.

Milvus Terms

Milvus Terminology

- Collection: A collection that consists of a set of entities and are equivalent to a table in an RDBMS.
- Segment: A data file that Milvus automatically creates by merging inserted data. A collection can contain multiple segments. One segment can contain multiple entities. During search, Milvus searches each segment, filters deleted data, and returns the combined result.
- Entity: A group of fields that correspond to real world objects. These fields can be structured data representing object properties or vectors representing object features.
- Entity ID: A guaranteed unique value that can be used to always reference an entity.

  Currently, Milvus does not support entity ID de-duplication and it is possible to have duplicate IDs in a segment.
- Field: A field within an entity. A field can either be structured data, such as numbers, strings, or unstructured data, such as vectors.
- Vector: A type of field representing the feature of an object.

  Currently, an entity can only contain up to one vector.
- Index: An index built based on raw data and improves the speed of data retrieval operations on a collection.
- Mapping: A set of rules that define how data is organized in a collection.

Quick Start

Installation Overview

Ensure that you have read Milvus Distributions and understood the differences in terms of performance and scenarios.

Docker is the recommended way to install and run Milvus.

- Install CPU-only Milvus on Docker
- Install GPU-enabled Milvus on Docker

If you have not set up a Docker environment, see Build Milvus from Source to learn how to build Milvus from source.

Install and Start Milvus

CPU-only MilvusGPU-enabled Milvus

Prerequisites

Operating System Requirements

Operating system	Supported versions
CentOS	7.5 or higher
Ubuntu LTS	18.04 or higher

Hardware Requirements

Component	Recommended configuration
CPU	Intel CPU Sandy Bridge or higher.
CPU Instruction Set	
RAM	8 GB or more (depends on the data volume)
Hard Drive	SATA 3.0 SSD or higher

Software Requirements

Software	Version
Docker	19.03 or higher

Please ensure that the available memory is greater than the sum of cache.insert_buffer_size and cache.cache_size set in the server_config.yaml file.

Confirm Docker Status

Confirm that the Docker daemon is running in the background:

```
$ sudo docker info
```

If you do not see the server listed, start the Docker daemon.

On Linux, Docker needs sudo privileges. To run Docker commands without sudo privileges, create a docker group and add your users (see Post-installation Steps for Linux for details).

Pull Docker Image

Pull the CPU-only image:

```
$ sudo docker pull milvusdb/milvus:1.1.0-cpu-d050721-5e559c
```

If you cannot use your host to acquire Docker images and configuration files online because of network restrictions, please acquire them online from another available host, save them as a TAR file, pass it on to your local machine, and then load the TAR file as a Docker image:

Click here to view the sample code.

Save the Docker image as a TAR file, and pass it on to your local machine:

$ docker save milvusdb/milvus > milvus_image.tar

Load the TAR file as a Docker image:

$ docker load < milvus_image.tar

If pulling the docker image is too slow or keeps failing, see Operational FAQ for solutions.

Download Configuration Files

```
$ mkdir -p /home/$USER/milvus/conf
$ cd /home/$USER/milvus/conf
$ wget https://raw.githubusercontent.com/milvus-io/milvus/v1.1.0/core/conf/demo/server_config.yaml
```

If you cannot download configuration files via the wget command, you can create a server_config.yaml file under /home/$USER/milvus/conf, and then copy the content from server config to it.

Start Docker Container

Start Docker container and map the paths to the local files to the container:

```
$ sudo docker run -d --name milvus_cpu_1.1.0 \
-p 19530:19530 \
-p 19121:19121 \
-v /home/$USER/milvus/db:/var/lib/milvus/db \
-v /home/$USER/milvus/conf:/var/lib/milvus/conf \
-v /home/$USER/milvus/logs:/var/lib/milvus/logs \
-v /home/$USER/milvus/wal:/var/lib/milvus/wal \
milvusdb/milvus:1.1.0-cpu-d050721-5e559c
```

The `docker run` options used in the above command are defined as follows:

- `-d`: Runs container in the background and prints container ID.
- `--name`: Assigns a name to the container.
- `-p`: Publishes a container' s port(s) to the host.
- `-v`: Mounts the directory into the container.

Confirm the running state of Milvus:

```
$ sudo docker ps
```

If the Milvus server does not start up properly, check the error logs:

```
$ sudo docker logs milvus_cpu_1.1.0
```

FAQ

Can I install Milvus on Windows?

Yes, so long as you have set up a Docker environment on your operating system.

Why does Milvus return Illegal instruction during startup?

If your CPU does not support SSE42, AVX, AVX2, or AVX512, Milvus cannot start properly. You can use cat /proc/cpuinfo to check the supported instruction sets.

How to migrate data in Milvus?

For details, see data migration.

Data formats of different versions may not be compatible with each other. The current data format is backward compatible with Milvus v0.7.0.

Is Docker the only way to install and run Milvus?

No. You can also build Milvus from source code in Linux. See Build Milvus from source code for more information.

How to set nlist or nprobe for IVF indexes?

In general terms, the recommended value of nlist is $4 \times \text{sqrt(n)}$, where n is the total number of entities in a segment.

Determining `nprobe` is a trade-off between search performance and accuracy, and based on your dataset and scenario. It is recommended to run several rounds of tests to determine the value of `nprobe`.

The following charts are from a test running on the sift50m dataset and IVF_SQ8 index. The test compares search performance and recall rate between different `nlist`/`nprobe` pairs.

We only show the results of GPU-enabled Milvus here, because the two distributions of Milvus show similar results.

Key takeaways: This test shows that the recall rate increases with the `nlist`/`nprobe` pair.

Key takeaways: When `nlist` is 4096 and `nprobe` 128, Milvus shows the best search performance.

What's next

- If you're just getting started with Milvus:
  - Try an example program
  - Learn basic Milvus operations
  - Try Milvus Bootcamp
- If you're ready to run Milvus in production:
  - Build a monitoring and alerting system to check real-time application performance.
  - Tune Milvus performance through configuration.
- If you want to use GPU-accelerated Milvus for search in large datasets:
  - Install GPU-enabled Milvus

Hello Milvus

After the Milvus server is successfully started, you can use this example program to create a table, insert 10 vectors, and then run a vector similarity search.

1. Make sure Python 3.6 and a compatible pip are installed.
2. Install Milvus Python SDK.

   ```
   ### Install Milvus Python SDK
   $ pip3 install pymilvus==1.1.0
   ```

   To learn more about Milvus Python SDK, go to Milvus Python SDK Readme.
3. Download Python example code.

   ```
   ### Download Python example
   $ wget https://raw.githubusercontent.com/milvus-io/pymilvus/v1.1.0/examples/example.py
   ```

   If you cannot use wget to download the example code, you can also create example.py and copy the example code.
4. Run the example code.

   ```
   ### Run Milvus Python example
   $ python3 example.py
   ```
5. Confirm the program is running correctly.

   ```
   Query result is correct.
   ```

Congratulations! You have successfully completed your first vector similarity search with Milvus.

What's next

- Learn basic Milvus operations in Milvus
- Try Milvus bootcamp to check more solutions

Connect to the Server

This article describes how to connect to a Milvus server from a Python client.

See Python API documentation for details about APIs.

We recommend using Milvus Sizing Tool to estimate the hardware resources required for the data.

1. Import pymilvus:

   ```
   ### Import pymilvus.
   >>> from milvus import Milvus, IndexType, MetricType, Status
   ```

2. Use any of the following methods to connect to the Milvus server:

```
### Connect to the Milvus server.
>>> milvus = Milvus(host='localhost', port='19530')
```

In the above code, host and port both use default values. You can change them to your IP address and port.

```
>>> milvus = Milvus(uri='tcp://localhost:19530')
```

FAQ

Does Milvus' Python SDK have a connection pool?

Python SDKs corresponding to Milvus v0.9.0 or later have a connection pool. There is no upper limit on the default number of connections in a connection pool.

How to fix the error when I install pymilvus on Windows?

Try installing pymilvus in a Conda environment.

Create/Drop a Collection

Create and Drop a Collection

This article provides Python sample codes for creating or dropping collections.

See Example Program for more detailed usage.

Create a Collection

1. Prepare the parameters needed to create the collection:

```
### Prepare collection parameters.
>>> param = {'collection_name':'test01', 'dimension':256, 'index_file_size':1024, 'metric_type':Metric
```

2. Create a collection named `test01`, with a dimension of 256 and an index file size of 1024 MB. It uses Euclidean distance (L2) as the distance measurement method.

```
### Create a collection.
>>> milvus.create_collection(param)
```

Drop a Collection

```
### Drop a collection.
>>> milvus.drop_collection(collection_name='test01')
```

FAQ

How can I get the best performance from Milvus through setting index_file_size?

You need to set index_file_size when creating a collection from a client. This parameter specifies the size of each segment, and its default value is 1024 in MB. When the size of newly inserted vectors reaches the specified volume, Milvus packs these vectors into a new segment. In other words, newly inserted vectors do not go into a segment until they grow to the specified volume. When it comes to creating indexes, Milvus creates one index file for each segment. When conducting a vector search, Milvus searches all index files one by one.

As a rule of thumb, we would see a 30% ~ 50% increase in the search performance after changing the value of index_file_size from 1024 to 2048. Note that an overly large index_file_size value may cause failure to load a segment into the memory or graphics memory. Suppose the graphics memory is 2 GB and index_file_size 3 GB, each segment is obviously too large.

In situations where vectors are not frequently inserted, we recommend setting the value of index_file_size to 1024 MB or 2048 MB. Otherwise, we recommend setting the value to 256 MB or 512 MB to keep unindexed files from getting too large.

See Performance Tuning > Index for more information.

Can I update index_file_size and metric_type after creating a collection?

No, you cannot.

Is there a limit on the total number of collections and partitions?

There is no limit on the number of collections. The upper limit on the number of partitions in a collection is 4096.

What is the maximum dimension of a vector in Milvus?

Milvus can support vectors with up to 32,768 dimensions.

Create/Drop a Partition

Create and Drop a Partition

This article provides Python sample codes for creating or dropping partitions.

Create a Partition

To improve search efficiency, you can divide a collection into several partitions by tags. In fact, each partition is a collection.

```
### Create a partition.
>>> milvus.create_partition('test01', 'tag01')
```

Drop a Partition

```
>>> milvus.drop_partition(collection_name='test01', partition_tag='tag01')
```

Insert/Delete Vectors

Insert and Delete Vectors

You can perform vector operations on collections or partitions. This article talks about the following topics:

- Insert vectors to a collection
- Insert vectors to a partition
- Delete vectors by ID

Insert Vectors to a Collection

1. Randomly generate 20 256-dimensional vectors:

   ```
   >>> import random
   ### Generate 20 vectors of 256 dimensions.
   >>> vectors = [[random.random() for _ in range(256)] for _ in range(20)]
   ```

2. Insert a list of vectors. If you do not specify vector IDs, Milvus automatically assigns IDs to the vectors.

   ```
   ### Insert vectors.
   >>> milvus.insert(collection_name='test01', records=vectors)
   ```

   You can also specify the vector IDs:

   ```
   >>> vector_ids = [id for id in range(20)]
   >>> milvus.insert(collection_name='test01', records=vectors, ids=vector_ids)
   ```

Insert Vectors to a Partition

```
>>> milvus.insert('test01', vectors, partition_tag="tag01")
```

Delete Vectors by ID

Suppose your collection contains the following vector IDs:

```
>>> ids = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

You can delete the vectors with the following command:

```
>>> milvus.delete_entity_by_id(collection_name='test01', id_array=ids)
```

After calling delete, you can call flush again to ensure that the newly inserted data is visible and the deleted data is no longer recoverable.

FAQ

Is there a length limit on the self-defined entity IDs?

Entity IDs must be non-negative 64-bit integers.

Can I insert vectors with existing IDs?

Yes, you can. If you insert vectors with an existing ID, you would end up having duplicate IDs.

Does Milvus support inserting while searching?

Yes.

Is there a volume limit on the vectors inserted each time?

Vectors inserted each time must not exceed 256 MB.

What is the maximum dimension of a vector in Milvus?

Milvus can support vectors with up to 32,768 dimensions.

Create/Drop an Index

Create and Drop an Index

This article provides Python sample codes for creating or dropping indexes.

Create an index

Currently, a collection only supports one index type. When you change the index type of a collection, Milvus automatically deletes the old index file. Before creating other indexes, a collection uses FLAT as the default index type.

create_index() specifies the index type of a collection and synchronously creates indexes for the previously inserted data. When the size of the subsequently inserted data reaches the index_file_size, Milvus automatically creates indexes in the background. For streaming data, it is recommended to create indexes before inserting the vector so that the system can automatically build indexes for the next data. For static data, it is recommended to import all the data at first and then create indexes. See Index Sample Program for details about using index.

1. Prepare the parameters needed to create indexes (take IVF_FLAT as an example). The index parameters are stored in a JSON string, which is represented by a dictionary in the Python SDK.

   ```
   ### Prepare index param.
   >>> ivf_param = {'nlist': 16384}
   ```

   Different index types requires different indexing parameters. They must all have a value.

2. Create index for the collection:

   ```
   ### Create an index.
   >>> milvus.create_index('test01', IndexType.IVF_FLAT, ivf_param)
   ```

Ensure you enable GPU when indexing and searching with IVF_SQ8H.

Drop an Index

After deleting the index, the collection uses the default index type FLAT again.

```
>>> milvus.drop_index('test01')
```

FAQ

How to set the value of nlist when I build indexes?

It depends on your scenario. See Performance tuning > Index for more information.

Can Milvus create different types of index for different partitions in the same collection?

No. A collection can have only one index type at a time.

Does Milvus create new indexes after vectors are inserted?

Yes. When the inserted vectors grow to a specified volume, Milvus creates a new segment and starts to create an index file for it at the same time. The building of the new index file does not affect the existing index files.

Conduct a vector search

Conduct a Vector Search

Milvus supports searching vectors in a collection or partition.

Search for Vectors in a Collection

1. Create search parameters. The search parameters are stored in a JSON string, which is represented by a dictionary in the Python SDK.

   ```
   >>> search_param = {'nprobe': 16}
   ```

   Different index types requires different search parameters. You must assign values to all search parameters. See Vector Indexes for more information.

2. Create random vectors as `query_records` to search:

   ```
   ### Create 5 vectors of 256 dimensions.
   >>> q_records = [[random.random() for _ in range(256)] for _ in range(5)]
   >>> milvus.search(collection_name='test01', query_records=q_records, top_k=2, params=search_param)
   ```

   top_k means searching the k vectors most similar to the target vector. It is defined during the search.

   The range of top_k is [1, 16384].

Search Vectors in a Partition

```
### Create 5 vectors of 256 dimensions.
>>> q_records = [[random.random() for _ in range(256)] for _ in range(5)]
>>> milvus.search(collection_name='test01', query_records=q_records, top_k=1, partition_tags=['tag01'], pa
```

If you do not specify partition_tags, Milvus searches similar vectors in the entire collection.

FAQ

Why is my recall rate unsatisfying?

You can increase the value of nprobe when searching from a client. The greater the value, the more accurate the result, and the more time it takes. See Performance Tuning > Index for more information.

Does Milvus support inserting while searching?

Yes.

Does the size of a collection affect vector searches in one of its partitions, especially when it holds up to 100 million vectors?

No. If you have specified partitions when conducting a vector search, Milvus searches the specified partitions only.

Does Milvus load the whole collection to the memory if I search only certain partitions in that collection?

No, Milvus only loads the partitions to search.

Are queries in segments processed in parallel?

Yes. But the parallelism processing mechanism varies with Milvus versions.

Suppose a collection has multiple segments, then when a query request comes in:

CPU-only Milvus processes the segment reading tasks and the segment searching tasks in pipeline.

On top of the abovementioned pipeline mechanism, GPU-enabled Milvus distributes the segments among the available GPUs.

See How Does Milvus Schedule Query Tasks for more information.

Will a batch query benefit from multi-threading?

If your batch query is on a small scale (nq < 64), Milvus combines the query requests, in which case multi-threading helps.

Otherwise, the resources are already exhausted, hence multi-threading does not help much.

Why the search is very slow?

Check if the value of cache.cache_size in server_config.yaml is greater than the size of the collection.

Why do I see a surge in memory usage when conducting a vector search immediately after an index is created?

This is because:

- Milvus loads the newly created index file to the memory for the vector search.
- The original vector files used to create the index are not yet released from the memory, because the size of original vector files and the index file has not exceeded the upper limit specified by cache.cache_size.

Why does the first search take a long time after Milvus restarts?

This is because, after restarting, Milvus needs to load data from the disk to the memory for the first vector search. You can set preload_collection in server_config.yaml and load as many collections as the memory permits. Milvus loads collections to the memory each time it restarts.

Otherwise, you can call load_collection() to load collections to the memory.

Data Flushing

When performing operations that change data, you can flush the data in the collection from memory to make the data available. Milvus also performs an automatic flush. The automatic flush function flushes all existing collection data every a fixed interval (1 second).

```
>>> milvus.flush(collection_name_array=['test01'])
```

After calling delete, you can call flush again to ensure that the newly inserted data is visible and the deleted data is no longer recoverable.

FAQ

Why my data cannot be searched immediately after insertion?

This is because the data has not been flushed from memory to disk. To ensure that data can be searched immediately after insertion, you can call flush. However, calling this method too often creates too many small files and affects search speed.

Compact Segments

Milvus automatically merges the inserted vector data into segments. A collection can contain multiple segments. After deleting some vector data in a segment, the system cannot automatically release the space occupied by the deleted vector data. So, you need to compact the segments in the collection to free up extra space.

```
>>> milvus.compact(collection_name='test01', timeout=1)
```

Close the Client

This article describes how to close a Python client.

```
>>> milvus.close()
```

Reference

Milvus Server Configuration

Configuration overview

The configurations apply to both Milvus Standalone and Cluster solutions.

Milvus file structure

After successfully starting Milvus server, you can see a Milvus folder at home/$USER/milvus, which contains the following files:

- milvus/db (database storage)
- milvus/logs (log storage)
- milvus/conf (configuration file folder)
  - server_config.yaml (server configuration)

Updating configurations

Editing the configuration file

You can directly edit the configuration file. You must restart Milvus every time a configuration file is updated.

```
$ docker restart <container_id>
```

Here we use Milvus' system configuration file server_config.yaml as an example to demonstrate how to modify the log level and log path:

```
logs:
  level: info
  path: /var/lib/milvus/logs
```

Updating configurations during runtime

You can update parameters in server_config.yaml from a Milvus client. See Client Reference for more information.

Changes to the following parameters take effect immediately without the need to restart Milvus.

- Section `cache`
  - `cache_size`
  - `insert_buffer_size`
- Section `gpu`
  - `enable`
  - `cache_size`
  - `gpu_search_threshold`
  - `search_devices`
  - `build_index_devices`

For other parameters, you still need to restart Milvus for the changes to take effect.

`server_config.yaml` parameters

Before changing these settings, welcome to consult Milvus team on GitHub issues or our Slack channel.

Section **`cluster`**

Parameter	Description	Type	Default
`enable`	Whether to enable cluster mode.	Boolean	`false`
`role`	Milvus deployment role:	Role	`rw`

Section **`general`**

Parameter	Description	Type	Default
`timezone`	Uses UTC-x or UTC+x to specify a time zone. For example, use UTC+8 to represent China Standard Time.	Timezone	`UTC+8`
`meta_uri`	URI for metadata storage, using SQLite (for single server Milvus) or MySQL (for distributed cluster Milvus). Format: `dialect://username:password@host:port/database`. `dialect` can be either `sqlite` or `mysql`. Replace the other fields with real values.	URI	`sqlite://:@:/`

Section **`network`**

Parameter	Description	Type	Default
`bind.address`	IP address that Milvus server monitors.	IP	`0.0.0.0`
`bind.port`	Port that Milvus server monitors. Range: [1025, 65534].	Integer	`19530`
`http.enable`	Whether to enable HTTP server.	Boolean	`true`
`http.port`	Port that Milvus HTTP server monitors. Range: [1025, 65534].	Integer	`19121`

Section **`storage`**

Parameter	Description	Type	Default
`path`	path to Milvus data files, including vector data files, index files, and the metadata.	Path	`/var/lib/milvus`
`auto_flush_interval`	The interval, in seconds, at which Milvus automatically flushes data to disk. Range: [0, 3600]. `0` means disabling the regular flush.	Integer	`1`

Section **`wal`**

Parameter	Description	Type	Default
`enable`	Whether to enable write-ahead logging (WAL) in Milvus. If enabled, Milvus writes all data changes to log files in advance before implementing data changes. WAL ensures the atomicity and durability for Milvus operations.	Boolean	true

Parameter	Description	Type	Default
`recovery_error_ignore`	Whether to ignore logs with errors that happens during WAL recovery.	Boolean	true
`buffer_size`	Total size of the read and write WAL buffer in Bytes. Range: 64MB ~ 4096MB. If the value you specified is out of range, Milvus automatically uses the boundary value closest to the specified value. It is recommended you set `buffer_size` to a value greater than the inserted data size of a single insert operation for better performance.	String	`256MB`
`wal_path`	path to WAL log files.	String	`/var/lib/milvus/w`

Section `cache`

Parameter	Description	Type	Default
`cache_size`	The size of the CPU memory for caching data for faster query. The sum of `cache_size` and `insert_buffer_size` must be less than the system memory size.	String	`4GB`
`insert_buffer_size`	Buffer size used for data insertion. The sum of `insert_buffer_size` and `cache_size` must be less than the system memory size.	String	`1GB`
`preload_collection`	A comma-separated list of collection names to load when Milvus server starts up.	StringList	N/A

Section `gpu`

This section determines whether to enable GPU support/usage in Milvus. GPU support, which uses both CPU and GPUs for optimized resource utilization, can achieve accelerated search performance on very large datasets.

Parameter	Description	Type	Default
`enable`	Whether to enable GPU usage in Milvus.	Boolean	`false`
`cache_size`	Size of the GPU memory for caching data. It must be less than the total memory size of the graphics card.	String	`1GB`
`gpu_search_threshold`	The threshold of GPU search. If `nq` represents the number of vectors to be searched for a single batch of queries, the search stragety is as follows:	Integer	`1000`
`search_devices`	A list of GPU devices used for search computation. Must be in format: `gpux`, where `x` is the GPU number, such as `gpu0`.	DeviceList	`gpu0`
`build_index_devices`	A list of GPU devices used for index building. Must be in format: `gpux`, where `x` is the GPU number, such as `gpu0`.	DeviceList	`gpu0`

In Milvus, index building and search computation are separate processes, which can be executed on CPU, GPU, or both. You can assign index building and search computation to multiple GPUs by adding GPUs under search_devices or build_index_devices. See the following YAML sample code:

```yaml
search_devices:
  - gpu0
  - gpu1
build_index_devices:
  - gpu0
  - gpu1
```

Section `logs`

Parameter	Description	Type	Default
`level`	Log level in Milvus. Log Levels: `debug` < `info` < `warning` < `error` < `fatal`.	String	`debug`
`trace.enable`	Whether to enable trace level logging.	Boolean	`true`
`path`	Absolute path to the folder holding the log files.	String	`/var/lib/milvus/l`
`max_log_file_size`	The maximum size of each log file. Range: 512MB ~ 4096MB.	String	`1024MB`
`log_rotate_num`	The maximum number of log files that Milvus keeps for each logging level. Range: [0, 1024]. `0` means that the number of stored log files does not have an upper limit.	Integer	`0`

Section `metric`

Parameter	Description	Type	Default
`enable`	Whether to enable the monitoring function of Prometheus.	Boolean	`false`
`address`	IP address of Prometheus Pushgateway.	IP	`127.0.0.1`
`port`	Port of Prometheus Pushgateway. Range: [1025, 65534].	Integer	`9091`

In the Milvus configuration file, space size should be written in the format of "number+unit", such as "4GB".

Do not add a space between the number and its unit.

The number must be an integer.

Available units include GB, MB, and KB.

FAQ

Besides the configuration file, how can I tell Milvus is using GPU for search?

Use any of the following methods:

Use nvidia-smi to monitor your GPU usage.

Use Prometheus to monitor performance metrics. See Visualize Metrics in Grafana > System performance metrics.

Check the Milvus server logs.

If I have set preload_collection, does Milvus service start only after all collections are loaded to the memory?

Yes. If you have set preload_collection in server_config.yaml, Milvus' service is not available until it loads all specified collections.

Why is my GPU always idle?

It is very likely that Milvus is using CPU for query. If you want to use GPU for query, you need to set the value of gpu_search_threshold in server_config.yaml to be less than nq (number of vectors per query).

You can use gpu_search_threshold to set the threshold: when nq is less than this value, Milvus uses CPU for queries; otherwise, Milvus uses GPU instead.

We do not recommend enabling GPU when the query number is small.

Why is the time in the log files different from the system time?

The log files in the Docker container use UTC time by default. If your host machine does not use UTC time, then the time in the log files is different. We recommend that you mount the log files onto your host machine to keep the time consistent between the log and the host.

Manage metadata with MySQL

Manage Metadata with MySQL

By default, Milvus uses SQLite for metadata management because it is easy to use, robust, and requires no additional services. However, we still recommend using MySQL in a production environment for improved reliability.

In CentOS, Milvus does not support MySQL 8.0 or higher.

Follow the steps below to use MySQL as metadata management service in Linux:

1. Pull the latest image of MySQL:

   ```
   $ docker pull mysql:5.7
   ```

2. Launch MySQL service. You can set your own password and port.

   ```
   $ docker run -p 3306:3306 -e MYSQL_ROOT_PASSWORD=123456 -d mysql:5.7
   ```

3. Use root account and the IP of the host that runs MySQL service (`<MySQL_server_host IP>`) to log in MySQL. Press <ENTER> to enter the password you set in the previous step.

   ```
   $ mysql -h<MySQL_server_host IP> -uroot -p
   ```

4. Enter MySQL client command line interface to create a database. Here we use `milvus` as the database name.

   ```
   mysql> create database milvus;
   ```

5. Quit MySQL client and update the `meta_uri` parameter in server_config.yaml. Use the IP of the host that runs MySQL service (`<MySQL_server_host IP>`). Note that the password, IP address, port, and database name must be consistent with your previous settings.

   ```
   meta_uri: mysql://root:123456@<MySQL_server_host IP>:3306/milvus
   ```

6. Use the updated server_config.yaml to launch Milvus.

FAQ

Why does Milvus return database is locked?

If you use SQLite to manage metadata, you receive this error message when write requests occur frequently. We recommend using MySQL for metadata management. See Manage Metadata with MySQL for more information.

Why can't I find vectors on SQLite or MySQL?

Milvus stores vectors and indexes directly in the disk as files, not in SQLite or MySQL. It uses SQLite or MySQL to store metadata of the vectors instead. See Storage Concepts for more information.

Can I use SQL Server or PostgreSQL to store metadata in Milvus?

No, we only support storing metadata using SQLite or MySQL.

Related blogs

From data import, data storage to data querying and scheduling, our blogs on Medium provide detailed insights into the data management mechanism of Milvus.

- Managing Data in Massive-Scale Vector Database
- Improvements of the Data File Cleanup Mechanism
- Viewing Metadata
- Fields in Metadata Tables
- How to Manage Data Files with Metadata

Monitoring and Alerting

Overview

Monitoring and Alerting

It is critical to actively monitor the overall performance of a system running in production, and to create alerting rules that promptly send notifications when there are events that require investigation or intervention.

Milvus uses the following monitoring and alerting solutions:

- Prometheus to store and monitor its metrics:
  - Prometheus server which scrapes and stores time series data.
  - Client libraries for instrumenting monitoring metrics.
  - Pushgateway to push metric data and ensure short-lived monitoring metrics, which may not be scraped in time, to be exposed to Prometheus.
  - Alertmanager for alert handling.
- Grafana, an open source platform for time-series analytics, to visualize various performance metrics.

Workflow

Milvus collects monitoring data and pushes it to Pushgateway. At the same time, the Prometheus server periodically pulls data from Pushgateway and saves it to its time-series database. The following graph shows how Prometheus works in Milvus:

Start up Prometheus

Configure and Start Prometheus

This page describes how to configure and start up Prometheus, and how to connect Alertmanager to Prometheus for metrics visualization and early warning purposes.

Install Prometheus

1. Download the Prometheus tarball for your operating system.
2. Go to the directory holding the Prometheus file, and ensure that Prometheus is properly installed:

   ```
   $ ./prometheus --version
   ```

   You can add the path to Prometheus to PATH. This makes it easy to start Prometheus from any shell.

Configure and start Prometheus

1. Start Pushgateway:

   ```
   ./pushgateway
   ```

   You must start Pushgateway before starting the Milvus Server.

2. Start the Prometheus monitor in server_config.yaml and set the address and port number of Pushgateway:

   ```
   metric:
     enable: true                  ### Set the value to true to enable the Prometheus monitor.
     address: <your_IP_address>    ### Set the IP address of Pushgateway.
     port: 9091                    ### Set the port number of Pushgateway.
   ```

   In the Kubernetes cluster, you need to set the server_config.yaml for each node to monitor.

3. Go to the Prometheus root directory, and download starter Prometheus configuration file for Milvus:

   ```
   $ wget https://raw.githubusercontent.com/milvus-io/docs/master/v1.1.0/assets/monitoring/prometheus.yml
   ```

4. Download starter alerting rules for Milvus to the Prometheus root directory:

```
wget -P rules https://raw.githubusercontent.com/milvus-io/docs/master/v1.1.0/assets/monitoring/alert_r
```

5. Edit the Prometheus configuration file according to your needs:

   - `global`: Configures parameters such as `scrape_interval` and `evaluation_interval`.

```
global:
  scrape_interval:     2s ### Set the crawl time interval to 2s.
  evaluation_interval: 2s ### Set the evaluation interval to 2s.
```

   - `alerting`: Sets the address and port of Alertmanager.

```
alerting:
alertmanagers:
- static_configs:
   - targets: ['localhost:9093']
```

   - `rule_files`: Specifies the file that defines the alerting rules.

```
rule_files:
   - "alert_rules.yml"
```

   - `scrape_configs`: Sets `job_name` and `targets` for scraping data.

```
scrape_configs:
- job_name: 'prometheus'
   static_configs:
   - targets: ['localhost:9090']

- job_name: 'pushgateway'
   honor_labels: true
   static_configs:
   - targets: ['localhost:9091']
```

   See Prometheus Configuration for more information about the configuration file of Prometheus.

6. Start Prometheus:

```
./prometheus --config.file=prometheus.yml
```

After starting up Prometheus, you can display and render on its interface the metrics that Milvus provides. See Milvus Metrics for more information.

Configure Alertmanager

Events to create alert rules

Proactively monitoring metrics contributes to identification of emerging issues. Creating alerting rules for events requiring immediate intervention is essential as well.

This section includes the most important events for which you must create alerting rules.

Server is down

- Rule: Send an alert when the Milvus server is down.
- How to detect: If the Milvus server is down, No Data is displayed for various metrics on the monitoring dashboard.

CPU/GPU temperature is too high

- Rule: Send an alert when the CPU/GPU temperature exceeds 80 degrees Celsius.
- How to detect: Check the metrics `CPU Temperature` and `GPU Temperature` on the monitoring dashboard.

Configuration steps

1. Download the latest Alertmanager tarball for your operating system.
2. Ensure that Alertmanager is properly installed:

   ```
   $ alertmanager --version
   ```

   You can add the path to Alertmanager to PATH. This makes it easy to start Alertmanager from any shell.
3. Create the Alertmanager configuration file to specify the desired receivers for notifications, and add it to Alertmanager root directory.
4. Start the Alertmanager server, with the `--config.file` flag pointing to the configuration file:

   ```
   alertmanager --config.file=alertmanager.yml
   ```
5. Use your browser to open http://<hostname of machine running alertmanager>:9093, and use the Alertmanager UI to define rules for muting alerts.

FAQ

How can I differentiate if I have multiple Milvus nodes connected to Pushgateway?

You can add a Prometheus instance in prometheus.yaml. Then Prometheus or Granafa will show the monitoring data, as well as the source node.

Visualize Metrics in Grafana

Configure and start Grafana

1. Start Grafana:

   ```
   $ docker run -i -p 3000:3000 grafana/grafana
   ```
2. Use your browser to open http://<hostname of machine running grafana>:3000 and log into the Grafana UI.

Grafana's default username and password are both admin. You can create a Grafana account of your own.

3. Add Prometheus as a data source.
4. In Grafana UI, click Configuration > Data Sources > Prometheus, and configure the data source as follows:

Field	Definition
Name	Prometheus
Default	`True`
URL	http://<hostname of machine running prometheus>:9090
Access	Browser

5. Download the starter Grafana dashboard for Milvus:
6. Add the dashboard to Grafana.

   prometheus.png

Display and Render Milvus Metrics

You can use Grafana dashboard to determine how to display and render Milvus metrics. See Milvus Metrics for more information.

Milvus Metrics

Milvus outputs detailed time-series metrics during runtime. You can use Prometheus, Grafana, or any visualization tool that you think appropriate to display and render the following metrics:

- Milvus Performance Metrics
- System Performance Metrics: Metrics relating to CPU/GPU usage, network traffic, and disk read speed.
- Hardware Storage Metrics: Metrics relating to data size, data files, and storage capacity.

Milvus performance metrics

Metric	Description
Insert per Second	Number of vectors that are inserted in a second. (Real-time display)
Queries per Minute	Number of queries that are run in a minute. (Real-time display)
Query Time per Vector	Average time to query one vector. Divide the query elapsed time by the number of queried vectors.
Query Service Level	Query service level = n_queries_completed_within_threshold1 / n_queries Generally, it is recommended to set 3 time periods - threshold1, threshold2, and threshold3, to track the query service level.
Uptime	How long Milvus has been running. (Minutes)

System performance metrics

Metric	Description
GPU Utilization	GPU utilization ratio (%).
GPU Memory Usage	GPU memory (GB) currently consumed by Milvus.
CPU Utilization	CPU utilization ratio (%). Divide the time that the server is busy by the total elapsed time.
Memory Usage	Memory (GB) currently consumed by Milvus.
Cache Utilization	Cache utilization ratio (%).
Network IO	Network IO read/write speed (GB/s).
Disk Read Speed	Disk read speed (GB/s).
Disk Write Speed	Disk write speed (GB/s).

Hardware storage metrics

Metric	Description
Data Size	Total amount (GB) of data stored in Milvus.
Total File	Number of data files currently stored in Milvus.

Performance Tuning

Performance tuning

Tune insertion performance

See Storage Operations for the basic procedure from inserting data to writing data to disk.

If the amount of data is less than the upper limit of a single insertion (256 MB), batch insertion is much more efficient than a single insertion.

The following parameters in the system configuration file have an impact on the insertion performance:

- `wal.enable`

This parameter is used to enable or disable the Write Ahead Log (WAL) function (enabled by default). The processes of inserting data when write ahead log is enabled or disabled are as follows:

- When write ahead log is enabled, the write ahead log module writes data to the disk, and then turns to the insert operation.
- When write ahead log is disabled, the data insertion speed is faster. The system directly writes the data to the mutable buffer in the memory and immediately turns to the insert operation.

`delete` operations are faster when write ahead log is enabled. We recommend that you enable write ahead log to ensure reliability of your data.

- `storage.auto_flush_interval`

This parameter (1 second by default) refers to the interval time of the data flushing task in the background. Increasing this value can reduce the number of segment merges, reduce disk I/O, and increase the throughput rate of insert operations.

Milvus cannot search for data that has not been flushed within this time interval.

Besides, the parameter `index_file_size`, which is used when creating collections, has an impact on the insertion performance. The value of this parameter is 1024 MB by default and 4096 MB at most. The larger the `index_file_size`, the more time it takes to merge data to the size set by this parameter, which affects the throughput rate of the insert operation. The smaller the parameter, the more data segments are generated. This may worsen query performance.

Besides software-level elements, network bandwidth and storage media also play a role in the insertion performance.

Tune query performance

Factors that affect query performance include hardware environment, system parameters, indexes, and query scale.

Hardware environment

- When CPU is used for calculations, query performance depends on the CPU's frequency, number of cores, and supported instruction set.

Milvus has better query performance on CPUs that support the AVX instruction set.

- When GPU is used for calculations, query performance depends on the GPU's parallel computing capabilities and transmission bandwidth.

System parameters

See Milvus server configuration for information about how to configure system parameters.

- `cache.cache_size`

This parameter (4 GB by default) refers to the size of the cache space used for resident query data. If the cache space is insufficient to hold the required data, the data will be temporarily loaded from the disk during the query, which seriously affects query performance. Therefore, `cache_size` should be greater than the amount of data required by the query.

- The data size of the floating-point original vector can be estimated by "total number of vectors × dimension × 4".
- The data size of the binary type original vector can be estimated by "total number of vectors × dimension ÷ 8".

After the indexes are created (FLAT is not included), the index files require additional disk space and the query only needs to load the index files.

- The data volume of the IVF_FLAT index is basically equal to the total data volume of its original vectors.
- The data volume of the IVF_SQ8 / IVF_SQ8H index is equivalent to 25% to 30% of the total data volume of the original vectors.
- The data volume of the IVF_PQ index changes according to its parameters, which is generally lower than 10% of the total data volume of the original vectors.

- The data volume of HNSW/RNSG/Annoy index is greater than the total data volume of the original vectors.

By calling get_collection_stats, you can get the total amount of data required to query a collection.

- `gpu.gpu_search_threshold`

In the GPU version, GPU is enabled for query when the number of target vectors is greater than or equals to the `gpu_search_threshold` (1000 by default).

The performance of GPU queries depends on GPU and the speed at which the CPU loads data to the graphic memory. The advantages of parallel computing with GPUs cannot be fully utilized when processing a small number of target vectors. Only when the number of target vectors reaches a certain threshold, the query performance on GPUs will be better than on CPUs. In practice, the ideal value of this parameter can be obtained based on experimental comparison.

- `gpu.resource_resources`

Specifies the GPU devices used for querying. For scenarios with a large number of query target vectors, using multiple GPUs can significantly improve query efficiency.

- `gpu.build_index_resources`

Specifies the GPU devices used for indexing. For scenarios where data insertion and querying are concurrent, you can use GPUs to build indexes to avoid the index building task competing for CPU resources with the query task.

Index

See Index Overview for the basic concepts of vector index.

To choose the right index, you need to trade off between multiple indicators such as storage space, query performance, and query recall rate.

- FLAT index

FLAT is a brute-force search for vectors. It has the slowest search speed, but has the highest recall rate (100%) and takes up the smallest amount of disk space.

As the number of target vectors increases, the time spent on using CPUs to perform FLAT queries increases linearly. On the other hand, using GPU to perform FLAT queries guarantees the high efficiency of batch queries and little effect on the query time from the increasing number of target vectors.

- IVF Indexes

IVF indexes include IVF_FLAT, IVF_SQ8 / IVF_SQ8H, and IVF_PQ. The IVF_SQ8 / IVF_SQ8H and IVF_PQ indexes perform lossy compression on vector data to reduce the disk space occupied by index files.

All IVF indexes have two parameters: `nlist` and `nprobe`. `nlist` is the indexing parameter, `nprobe` the searching parameter. For more information about the recommended values, see Performance FAQ > How to set `nlist` and `nprobe` for IVF indexes?.

The following section provides formulae for estimating the calculation amount for queries on IVF indexes:

- The amount of calculation of a single segment = the number of target vectors × (`nlist` + (the number of vectors in a segment ÷ `nlist`) × `nprobe`)
- The number of segments = the total amount of aggregate data ÷ `index_file_size`
- The total amount of calculation of a collection = the amount of calculation of a single segment × the segment number

The larger the estimated total amount of calculation, the longer the query takes. In practice, you can get reasonable parameters through the above formulas, which provides high query performance under the premise of an acceptable recall rate.

In scenario with continuous data insertion, because Milvus does not index segments with a size less than index_file_size, it uses brute-force search as the query method. The amount of calculation can be estimated by multiplying the number of target vectors by the total number of segment vectors.

- HNSW / RNSG / Annoy index

The index parameters of HNSW, RNSG, and Annoy have a more complex impact on query performance. For more information, see Index Introduction.

Other

- Result set

The size of the result set depends on the number of target vectors and `topk`. The size of `topk` has little effect on the calculation. However, when the number of target vectors and `topk` are large, the time spent on serializing the result set and network transmission will increase accordingly.

- MySQL

Milvus uses MySQL as a Metadata backend service. When querying data, Milvus accesses MySQL multiple times to obtain Metadata information. Therefore, the response speed of the MySQL service greatly influences the query performance of Milvus.

- Preload

When querying data for the first time, the system needs to read the data from the disk and write the data to the cache. This is time-consuming. To avoid loading data during the first query, you can call the `load_collection` API in advance, or use the system parameter `preload_collection` to specify the segment to preload when starting Milvus.

- Compact segments

To filter deleted entities, Milvus reads delete_docs into memory when querying data. You call `compact` to clean up deleted entities and reduce filtering operations, thereby improving query performance.

Optimize storage

- Compact segments

Deleted entities do not participate in the calculation and takes up disk space. If a large number of entities have been deleted, you can call `compact` to free up disk space.

FAQ

Why is my GPU always idle?

It is very likely that Milvus is using CPU for query. If you want to use GPU for query, you need to set the value of gpu_search_threshold in server_config.yaml to be less than nq (number of vectors per query).

You can use gpu_search_threshold to set the threshold: when nq is less than this value, Milvus uses CPU for queries; otherwise, Milvus uses GPU instead.

We do not recommend enabling GPU when the query number is small.

Why the search is very slow?

Check if the value of cache.cache_size in server_config.yaml is greater than the size of the collection.

How can I get the best performance from Milvus through setting index_file_size?

You need to set index_file_size when creating a collection from a client. This parameter specifies the size of each segment, and its default value is 1024 in MB. When the size of newly inserted vectors reaches the specified volume, Milvus packs these vectors into a new segment. In other words, newly inserted vectors do not go into a segment until they grow to the specified volume. When it comes to creating indexes, Milvus creates one index file for each segment. When conducting a vector search, Milvus searches all index files one by one.

As a rule of thumb, we would see a 30% ~ 50% increase in the search performance after changing the value of index_file_size from 1024 to 2048. Note that an overly large index_file_size value may cause failure to load a segment into the memory or graphics memory. Suppose the graphics memory is 2 GB and index_file_size 3 GB, each segment is obviously too large.

In situations where vectors are not frequently inserted, we recommend setting the value of index_file_size to 1024 MB or 2048 MB. Otherwise, we recommend setting the value to 256 MB or 512 MB to keep unindexed files from getting too large.

See Performance Tuning > Index for more information.

Why GPU-enabled query is sometimes slower than CPU-only query?

Generally speaking, CPU-only query works for situations where nq (number of vectors per query) is small, whilst GPU-enabled query works best with a large nq, say 500.

Milvus needs to load data from the memory to the graphics memory for a GPU-enabled query. Only when the load time is negligible compared to the time to query, is GPU-enabled query faster.

Why sometimes the query time for a small dataset is longer?

If the size of the dataset is smaller than the value of index_file_size that you set when creating a collection, Milvus does not create an index for this dataset. Therefore, the time to query in a small dataset may be longer. You may as well call create_index to build the index.

Mishards: Cluster Sharding Middleware

What is Mishards

Mishards is a Milvus cluster sharding middleware developed in Python. It handles request forwarding, read-write separation, horizontal and dynamic scaling, providing you with additional capabilities in terms of expanded memory and computing power.

How Mishards works

Mishards cascades a request from upstream down to its sub-modules splitting the upstream request, and then collects and returns the results of the sub-services to upstream.

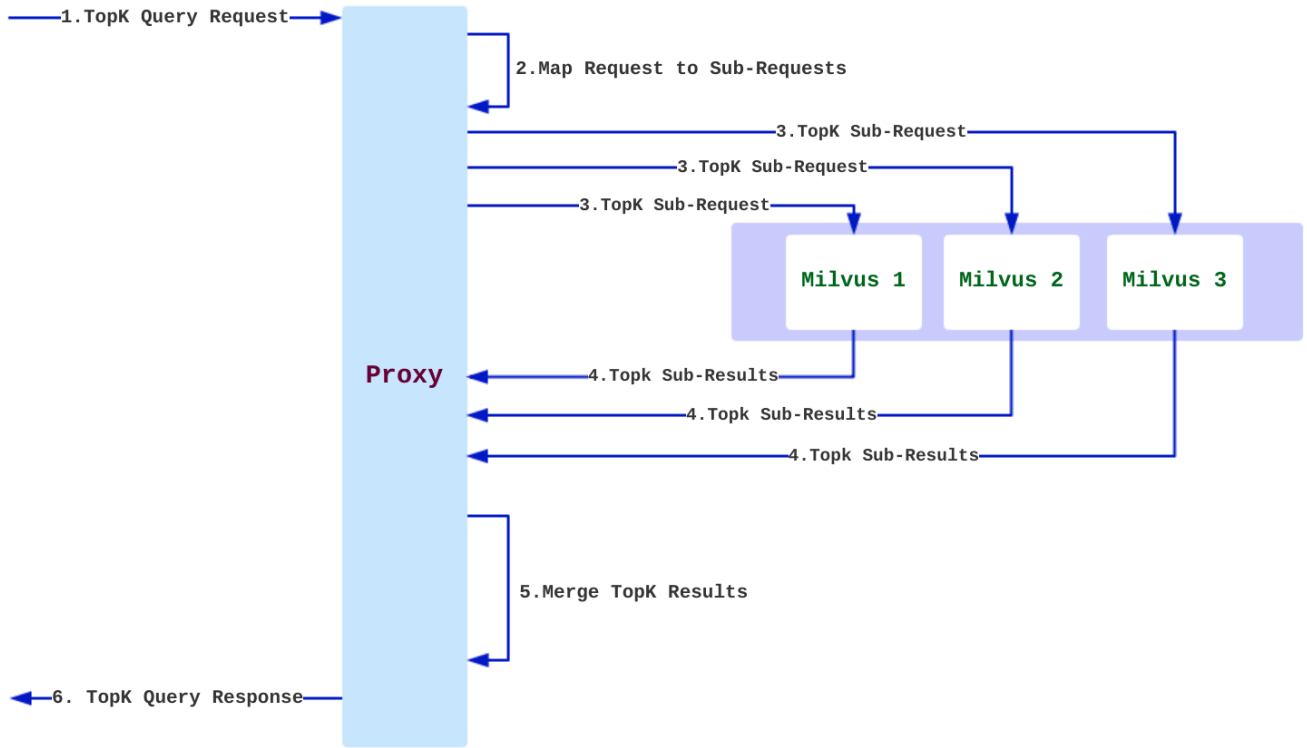

Figure 16: proxy

Target scenarios

Scenarios	Concurrency	Latency	Data scale	Suitable for Mishards
1	Low	Low	Medium / Small	No
2	High	Low	Medium / Small	No
3	Low	High	Large	Yes
4	Low	Low	Large	Yes
5	High	Low	Large	Yes

Mishards is suitable for scenarios with large data scale. So how to judge the size of the data scale? There is no standard answer to this question because it depends on the hardware resources used in the actual production environment. Here is a simple way to determine the size of the data scale:

1. If you do not care about latency, you can assume that a scenario has a large data scale when its data size is larger than the available capacity of the hard disk on a single server. For example, the calculation time of the server to batch process 5000 query requests is greater than the time to load data from the hard disk to the memory, so the available hard disk is the criteria for determining the data scale.
2. Otherwise, you can assume that a scenario has a large data scale when its data size is larger than the available memory on a single server.

Mishards-based cluster solution

Overall architecture

Main components

- Service discovery: Obtains the service addresses of the read and write nodes.
- Load balancer
- Mishards node: A stateless and scalable node.
- Milvus write node: An unscalable node. To avoid failure at a single node, you need to deploy a high availability strategy.
- Milvus read node: A stateful and scalable node.
- Shared storage service: Milvus read and write nodes share data through the shared storage service. Available options include NAS and NFS.
- Metadata service: Milvus supports only MySQL in the production environment.

Mishards configurations

Global configurations

Parameter	Required	Type	Default	Description
`Debug`	No	Boolean	`True`	Whether to enable the debug mode. Debug mode only affects the log level for now.
`TIMEZONE`	No	String	`UTC`	The time zone.
`SERVER_PORT`	No	Integer	`19530`	Defines the service port of Mishards.
`WOSERVER`	Yes	String		The address of Milvus write node. Format: `tcp://127.0.0.1:19530`

Metadata

Metadata records the structure information of the underlying data. In a distributed system, Milvus write nodes are the only producers of metadata; Mishards nodes, Milvus write nodes, and Milvus read nodes are consumers of Metadata.

Milvus only supports MySQL or SQLite as its Metadata backend for now. In a distributed system, the storage backend for metadata can only be MySQL.

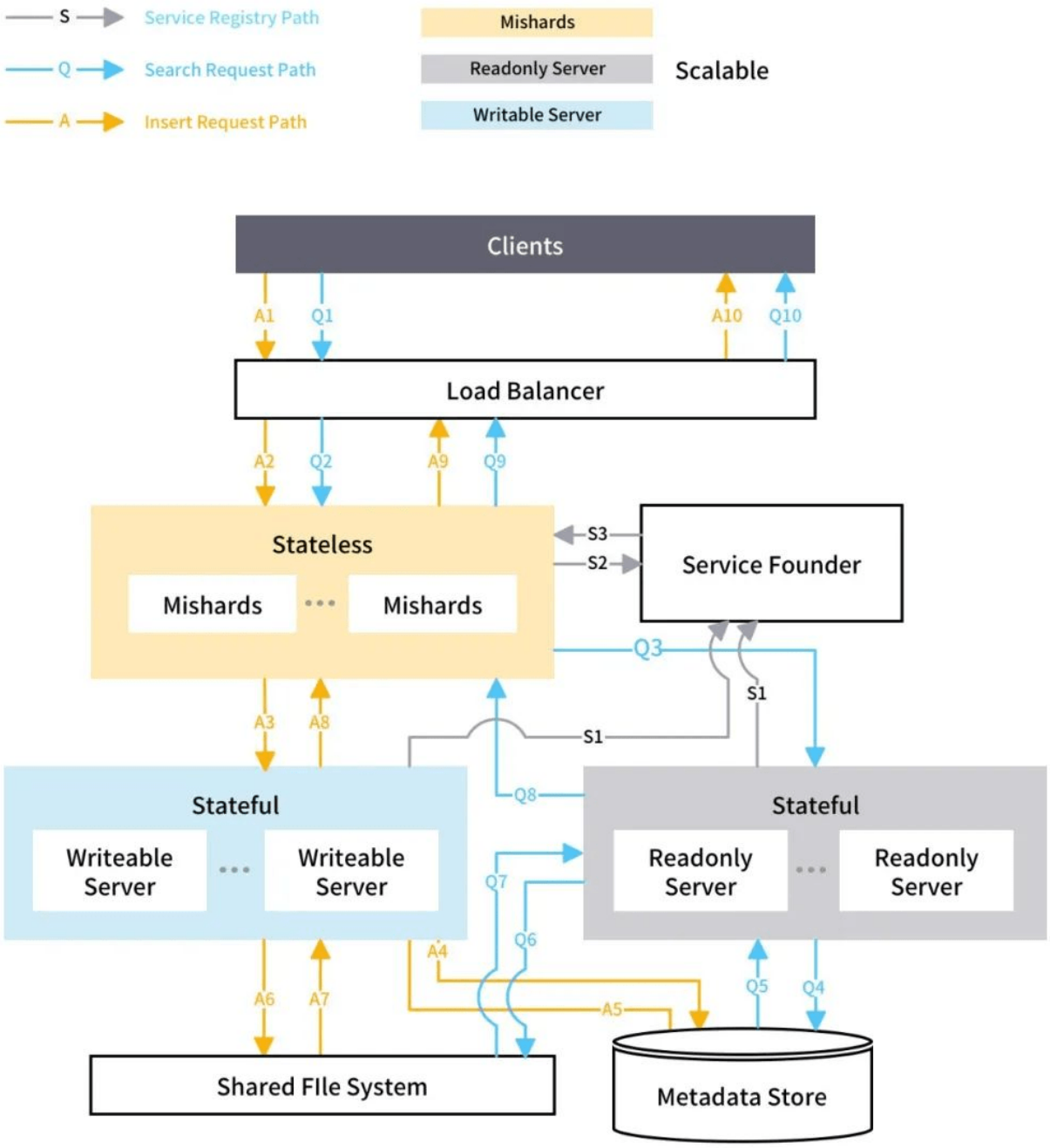

Figure 17: structure

Parameter	Required	Type	Default	Description
SQLALCHEMY_DATABASE_URI	Yes	String		Defines the address of the metadata storage database. The format conforms to the RFC-738-style, for example, mysql+pymysql://root:root@127.0.0.1:3306/milvus?charset=u
SQL_ECHO	No	Boolean	False	Whether to print detailed SQL queries.

Service discovery

Service discovery provides Mishards with the address information of all Milvus read and write nodes. Mishards defines the relevant service discovery API IServiceRegistryProvider, and provides extensions in extension mode. Milvus provides two extensions by default: KubernetesProvider corresponds to Kubernetes cluster; StaticProvider corresponds to static configuration. You can customize your own service discovery extension based on these two extensions.

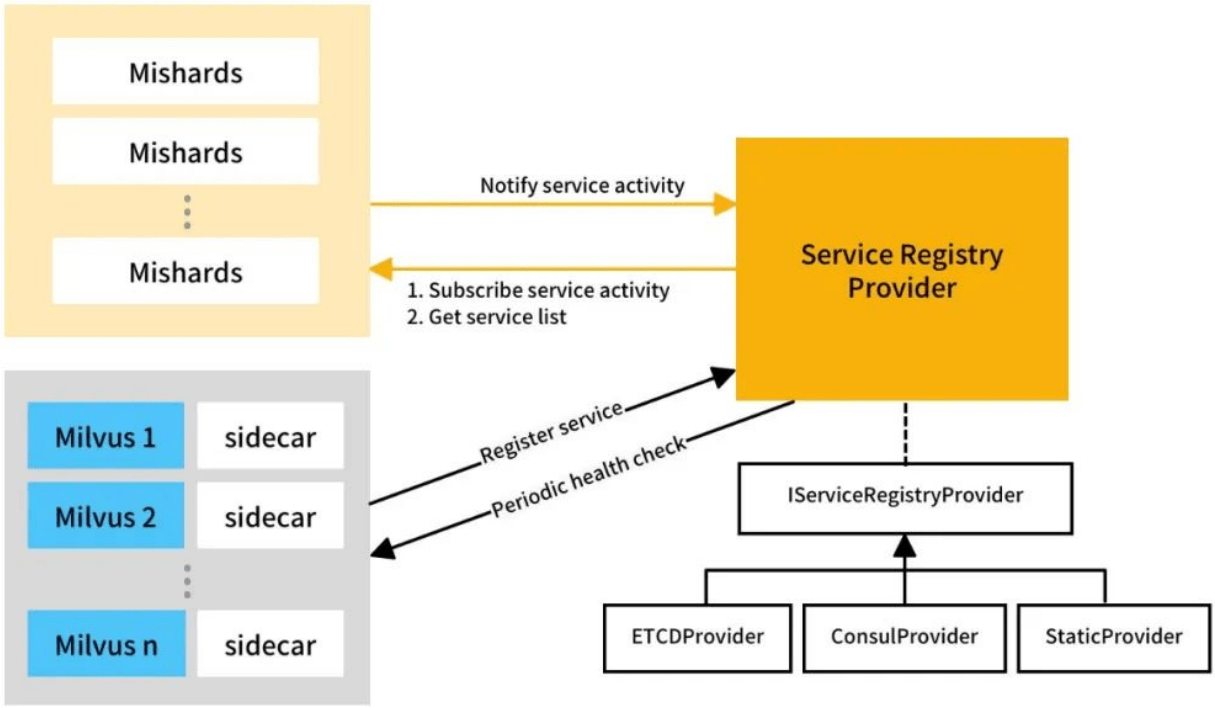

Figure 18: discovery

Parameter	Required	Type	Default	Description
DISCOVERY_STATIC_HOSTS	No	List	[]	When DISCOVERY_CLASS_NAME is static, defines the service address list. The addresses in the list are separated by comma, for example, 192.168.1.188,192.168.1.190.
DISCOVERY_STATIC_PORT	No	Integer	19530	When DISCOVERY_CLASS_NAME is static, defines the service address listening port.
DISCOVERY_PLUGIN_PATH	No	String		The search path to the customized service discovery extension (uses the system search path by default).

Parameter	Required	Type	Default	Description
DISCOVERY_CLASS_NAME	No	String	static	In the extension search path, searches for the class based on its name and instantiates it. At present, the system provides two classes: static (default) and kubernetes.
DISCOVERY_KUBERNETES_NAMESPACE	No	String		When DISCOVERY_CLASS_NAME is kubernetes, defines the namespace of the Milvus cluster.
DISCOVERY_KUBERNETES_IN_CLUSTER	No	Boolean	False	When DISCOVERY_CLASS_NAME is kubernetes, decides whether to run service discovery in the cluster.
DISCOVERY_KUBERNETES_POLL_INTERVAL	No	Integer	5	When DISCOVERY_CLASS_NAME is kubernetes, defines the monitoring period of the service discovery (unit: seconds).
DISCOVERY_KUBERNETES_POD_PATT	No	String		When DISCOVERY_CLASS_NAME is kubernetes, matches the regular expression to the name of Milvus Pod.
DISCOVERY_KUBERNETES_LABEL_SELECTOR		String		When SD_PROVIDER is kubernetes, matches the label of Milvus Pod, for example, tier=ro-servers.

Chain tracking

A distributed systems often distributes requests to multiple internal services. To facilitate troubleshooting, we need to track the call chains of internal services. The higher the complexity of the system, the more obvious the benefits of a viable chain tracking system. We choose OpenTracing, which is a distributed tracing standard that has entered CNCF. It provides APIs independent of the platform or vendor to facilitate implementation of a chain tracking system.

Mishards defines the chain tracking APIs and provides extensions in extension mode. It provides Jaeger-based extensions for now.

See Jaeger Doc to learn how to integrate Jaeger.

Parameter	Required	Type	Default	Description
TRACER_PLUGIN_PATH	No	String		The search path to the custom chain tracking extension (uses the system search path by default).
TRACER_CLASS_NAME	No	String		In the extension search path, searches for the class based on its name and instantiates it. Currently, only Jaeger is supported, but it is not used by default.
TRACING_SERVICE_NAME	No	String	mishards	When TRACING_CLASS_NAME is Jaeger, specifies the chain tracking service.
TRACING_SAMPLER_TYPE	No	String	const	When TRACING_CLASS_NAME is Jaeger, specifies the sampling type for chain tracking.
TRACING_SAMPLER_PARAM	No	Integer	1	When TRACING_CLASS_NAME is Jaeger, specifies the sampling frequency for chain tracking.
TRACING_LOG_PAYLOAD	No	Boolean	False	When TRACING_CLASS_NAME is Jaeger, decides whether to capture the payload for the chain tracking.

Log

The log files of the cluster service are distributed on different nodes, so you need to log in to the relevant server to obtain log files for troubleshooting. It is recommended that you use ELK log analysis component to collaboratively analyze multiple log files and troubleshoot problems.

Parameter	Required	Type	Default	Description
`LOG_LEVEL`	No	String	`DEBUG`	Log levels: `DEBUG` < `INFO` < `WARNING` < `ERROR`.
`LOG_PATH`	No	String	`/tmp/mishards`	Path to log files.
`LOG_NAME`	No	String	`logfile`	Name of log files.

Route

Mishards obtains the addresses of Milvus read and write nodes from the service discovery center and obtains the underlying Metadata information through the Metadata service. Its routing strategy is to consume these materials. As shown in the figure, there are 10 data segments (s1, s2, s3, ···, s10). We select a consistent hash routing strategy based on the name of data segments (`FileNameHashRingBased`). Mishards routes requests about s1, s4, s6, and s9 to the Milvus 1 node, routes requests about s2, s3, and s5 to the Milvus 2 node, and routes requests about s7, s8, and s10 to the Milvus 3 node.

Mishards defines APIs related to routing strategies and provides relevant extensions. You can customize your routes according to your business scenario and based on the default consistent hash routing extension.

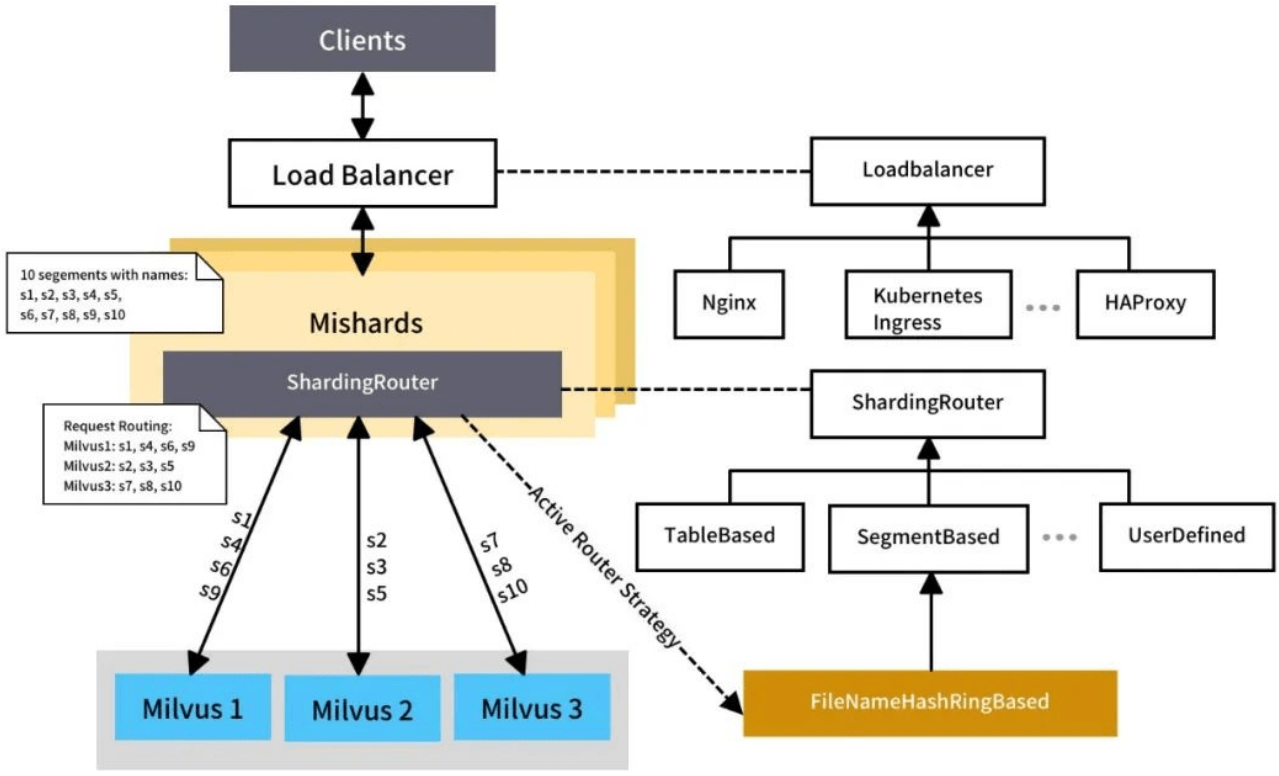

Figure 19: router

Parameter	Required	Type	Default	Description
`ROUTER_CLASS_NAME`	No	String	`FileBasedHashRingRouter`	In the extension search path, searches for the routed class based on the class name and instantiates it. Currently, the system only provides a consistent hash routing strategy `FileBasedHashRingRouter` based on the data segment name.
`ROUTER_PLUGIN_PATH`	No	String		The search path to the custom routing extensions (uses the system search path by default).

Mishards examples

Start Mishards

Prerequisites

- Milvus properly installed
- Python 3.6 or higher

Start a Milvus and Mishards instance

Follow these steps to start a Milvus instance and Mishards service on a machine:

1. Clone the Milvus repository to your local machine:

   ```
   $ git clone https://github.com/milvus-io/milvus -b 1.1
   ```

2. Install dependencies for Mishards:

   ```
   $ cd milvus/shards
   $ pip install -r requirements.txt
   ```

3. Start the Milvus service:

   - If your Docker version is earlier than v19.03:

   ```
   $ sudo docker  run --runtime=nvidia --rm -d -p 19530:19530 -v /tmp/milvus/db:/var/lib/milvus/db milvus
   ```

   - Otherwise:

   ```
   $ sudo docker run --gpus all --rm -d -p 19530:19530 -v /tmp/milvus/db:/var/lib/milvus/db milvusdb/milv
   ```

4. Change the directory permission:

   ```
   $ sudo chown -R $USER:$USER /tmp/milvus
   ```

5. Configure the environment variable for Mishards:

   ```
   $ cp mishards/.env.example mishards/.env
   ```

6. Start the Mishards service:

   ```
   $ python mishards/main.py
   ```

Start Mishards with docker-compose

`all_in_one` uses a Docker container to start 2 Milvus instances, 1 Mishards middleware instance, and 1 Jaeger chain tracking instance.

1. Install Docker Compose.
2. Clone the Milvus repository to the local machine:

   ```
   $ git clone https://github.com/milvus-io/milvus -b 1.1
   $ cd milvus/shards
   ```

3. Start all services:

   ```
   $ make deploy
   ```

4. Check the service status:

   ```
   $ make probe_deploy
   Pass ==> Pass: Connected
   Fail ==> Error: Fail connecting to server on 127.0.0.1:19530. Timeout
   ```

To view the service chain, open Jaeger Page in your browser.

To clean up all services:

```
$ make clean_deploy
```

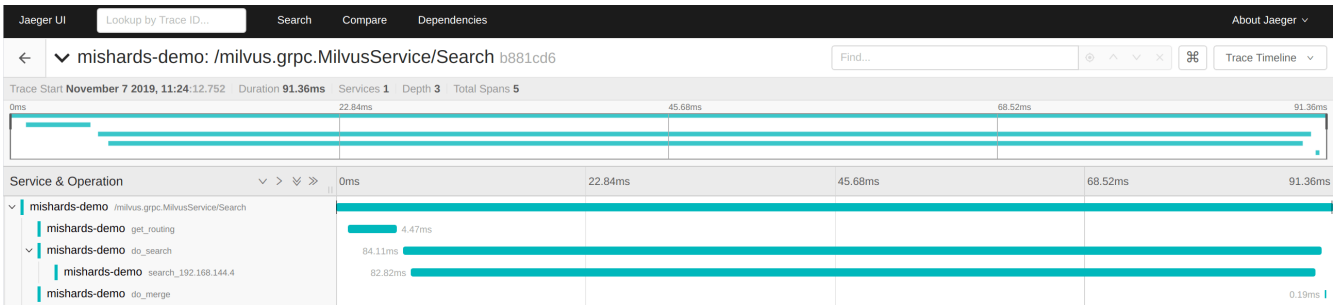

Figure 20: jaegerui

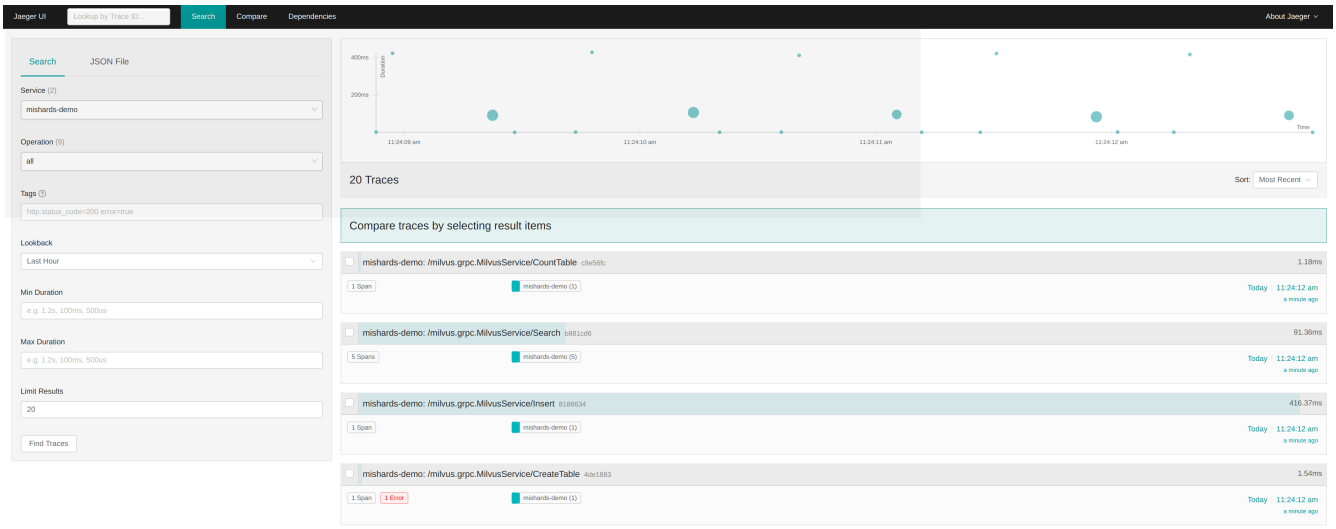

Figure 21: jaegertraces

Deploy Mishards cluster in Kubernetes

Prerequisites

- Kubernetes 1.10 or later
- Helm 2.12.0 or later

See Helm Docs for more information about using Helm.

Install Mishards

1. Add the Helm Chart repository:

   ```
   $ helm repo add stable https://kubernetes-charts.storage.googleapis.com
   ```

2. Install dependent libraries for Chart:

   ```
   $ git clone https://github.com/milvus-io/milvus-helm.git
   $ cd milvus-helm/charts/milvus
   $ helm dep update
   ```

3. Deploy Mishards:

   ```
   $ helm install --set cluster.enabled=true --set persistence.enabled=true milvus-release  .
   ```

4. Check the deployment status:

   ```
   $ helm list -f "milvus-release"
   ```

Uninstall Mishards

- Use Helm v2.x to uninstall Mishards:

  ```
  $ helm delete milvus-release
  ```

- Use Helm v3.x to uninstall Mishards:

  ```
  $ helm uninstall milvus-release
  ```

Upgrade from standalone service to Mishards cluster

Milvus-Helm supports upgrading from standalone service to Mishards cluster.

1. Deploy a standalone version of Milvus:

   ```
   $ helm install --set persistence.enabled=true milvus-release .
   ```

2. Upgrade to Mishards cluster:

   ```
   $ helm upgrade --set cluster.enabled=true --set persistence.enabled=true milvus-release .
   ```

Notes

Mishards is based on shared storage, so the Kubernetes cluster must have available Persistent Volumes (PV). Also, ensure that the PV can be used by multiple pods at the same time. You can enable Persistent Volumes by setting `persistence.enabled`.

1. In order to share data, the PV access mode must be set to `ReadOnlyMany` or `ReadWriteMany`.
2. Choose a file storage system:
   - If your cluster is deployed on AWS, use Elastic File System (EFS).
   - If your cluster is deployed on Azure, use Azure File Storage (AFS).

See Persistent Volumes for more information about applying for and managing Persistent Volume.

See Access Modes for more information about the access modes of Persistent Volume.

Usage

You can find all parameters supported by Milvus-Helm at Milvus Helm Charts.

1. Configure a cluster with multiple read nodes and multiple Mishards sharding middleware.

   Usually, we configure multiple nodes to ensure service availability and increase throughput rate. In the following example, the Mishards cluster includes 2 sharding middleware, 2 read nodes, and 1 write node.

   ```
   $ helm install
      --set cluster.enabled=true     \
      --set persistence.enabled=true \
      --set mishards.replicas=2       \
      --set readonly.replicas=2       \
      milvus-release .
   ```

   Here, the number of replica sets is controlled by `mishards.replicas` and `readonly.replicas`. Their default values are 1.

   Currently, the write nodes in the Mishards cluster cannot be expanded.

2. Use an externally configured MySQL cluster as the Metadata database.

   Sometimes the support for external MySQL is needed to cooperate with local deployment. Although Milvus-Helm's internal MySQL service does not guarantee high availability, you can increase availability through an external MySQL cluster. The following example shows the deployment based on external MySQL.

   ```
   $ helm install
      --set cluster.enabled=true             \
      --set persistence.enabled=true         \
      --set mysql.enabled=false              \
      --set externalMysql.enabled=true       \
      --set externalMysql.ip=192.168.1.xx    \
      --set externalMysql.port=3306          \
      --set externalMysql.user=root          \
      --set externalMysql.password=root      \
      --set externalMysql.database=milvus    \
      milvus-release .
   ```

   When using external MySQL, you do not need the built-in MySQL service of Helm. Therefore, you can disable the built-in MySQL service of Helm by setting `mysql.enabled=false`.

3. The read and write nodes of Milvus have different configurations.

   To reasonably use resources, we hope that the read nodes and the write nodes have different configurations. In the following example, we configure a read node with 16 GB memory and a write node with 8 GB memory.

   ```
   $ helm install
      --set cluster.enabled=true                    \
      --set persistence.enabled=true                \
      --set cache.cpuCacheCapacity=8                \
      --set readonly.cache.cpuCacheCapacity=16      \
      milvus-release .
   ```

   See Milvus configuration for more Milvus configuration parameters.

   See Milvus Helm Charts for more Milvus-Helm configuration parameters.

4. Configure the GPU resources.

   The use of GPU can effectively improve Milvus performance. In the following example, we allow write nodes to use GPU resources by setting `gpu.enabled=true` and prevent the read nodes from using GPU resources by setting `readonly.gpu.enabled=false`.

```
$ helm install
   --set cluster.enabled=true            \
   --set persistence.enabled=true        \
   --set gpu.enabled=true                \
   --set readonly.gpu.enabled=false      \
   milvus-release .
```

See Schedule GPUs for GPU resource management and scheduling in Kubernetes.

FAQ

Performance FAQ

Why does the first search take a long time after Milvus restarts?

This is because, after restarting, Milvus needs to load data from the disk to the memory for the first vector search. You can set `preload_collection` in server_config.yaml and load as many collections as the memory permits. Milvus loads collections to the memory each time it restarts.

Otherwise, you can call `load_collection()` to load collections to the memory.

Why the search is very slow?

Check if the value of `cache.cache_size` in server_config.yaml is greater than the size of the collection.

How do I improve Milvus' performance?

- Ensure that the value of `cache.cache_size` in server_config.yaml is greater than the size of the collection.
- Ensure that all segments are indexed.
- Check if there are other processes on the server consuming CPU resources.
- Adjust the values of `index_file_size` and `nlist`.
- If the search performance is unstable, you can add `-e OMP_NUM_THREADS=NUM` when starting up Milvus, where `NUM` is 2/3 of the number of CPU cores.

See Performance tuning for more information.

How to set **`nlist`** and **`nprobe`** for IVF indexes?

In general terms, the recommended value of nlist is $4 \times \text{sqrt(n)}$, where n is the total number of entities in a segment.

Determining `nprobe` is a trade-off between search performance and accuracy, and based on your dataset and scenario. It is recommended to run several rounds of tests to determine the value of `nprobe`.

The following charts are from a test running on the sift50m dataset and IVF_SQ8 index. The test compares search performance and recall rate between different `nlist`/`nprobe` pairs.

We only show the results of GPU-enabled Milvus here, because the two distributions of Milvus show similar results.

Key takeaways: This test shows that the recall rate increases with the `nlist`/`nprobe` pair.

Key takeaways: When `nlist` is 4096 and `nprobe` 128, Milvus shows the best search performance.

Why sometimes the query time for a small dataset is longer?

If the size of the dataset is smaller than the value of `index_file_size` that you set when creating a collection, Milvus does not create an index for this dataset. Therefore, the time to query in a small dataset may be longer. You may as well call `create_index` to build the index.

Why is my GPU always idle?

It is very likely that Milvus is using CPU for query. If you want to use GPU for query, you need to set the value of gpu_search_threshold in server_config.yaml to be less than nq (number of vectors per query).

You can use gpu_search_threshold to set the threshold: when nq is less than this value, Milvus uses CPU for queries; otherwise, Milvus uses GPU instead.

We do not recommend enabling GPU when the query number is small.

Why my data cannot be searched immediately after insertion?

This is because the data has not been flushed from memory to disk. To ensure that data can be searched immediately after insertion, you can call `flush`. However, calling this method too often creates too many small files and affects search speed.

Why does my CPU usage stay low?

Milvus processes queries in parallel. An `nq` less than 100 and data on a smaller scale do not require high level of parallelism, hence the CPU usage stays low.

How can I get the best performance from Milvus through setting `index_file_size`?

You need to set `index_file_size` when creating a collection from a client. This parameter specifies the size of each segment, and its default value is `1024` in MB. When the size of newly inserted vectors reaches the specified volume, Milvus packs these vectors into a new segment. In other words, newly inserted vectors do not go into a segment until they grow to the specified volume. When it comes to creating indexes, Milvus creates one index file for each segment. When conducting a vector search, Milvus searches all index files one by one.

As a rule of thumb, we would see a 30% ~ 50% increase in the search performance after changing the value of `index_file_size` from 1024 to 2048. Note that an overly large `index_file_size` value may cause failure to load a segment into the memory or graphics memory. Suppose the graphics memory is 2 GB and `index_file_size` 3 GB, each segment is obviously too large.

In situations where vectors are not frequently inserted, we recommend setting the value of `index_file_size` to 1024 MB or 2048 MB. Otherwise, we recommend setting the value to 256 MB or 512 MB to keep unindexed files from getting too large.

See Performance Tuning > Index for more information.

What is the importing performance of Milvus in practical terms?

When the client and the server are running on the same physical machine, it takes about 0.8 second to import 100,000 128-dimensional vectors (to an SSD disk). More specifically, the performance depends on the I/O speed of your disk.

Does searching while inserting affect the search speed?

- If the newly inserted vectors have not grown to the specified volume to trigger index creation, Milvus needs to load these data directly from disk to memory for a vector search.
- As of v0.9.0, if Milvus has started creating indexes for the newly inserted vectors, an incoming vector search interrupts the index creation process, causing a delay of about one second.

Will a batch query benefit from multi-threading?

If your batch query is on a small scale (`nq` < 64), Milvus combines the query requests, in which case multi-threading helps.

Otherwise, the resources are already exhausted, hence multi-threading does not help much.

Why GPU-enabled query is sometimes slower than CPU-only query?

Generally speaking, CPU-only query works for situations where `nq` (number of vectors per query) is small, whilst GPU-enabled query works best with a large `nq`, say 500.

Milvus needs to load data from the memory to the graphics memory for a GPU-enabled query. Only when the load time is negligible compared to the time to query, is GPU-enabled query faster.

Still have questions?

You can:

- Check out Milvus on GitHub. You're welcome to raise questions, share ideas, and help others.
- Join our Slack community to find more help and have fun!

Product FAQ

Is Milvus free of charge?

Milvus is an open-source project, and hence is free-of-charge.

Please adhere to Apache License 2.0, when using Milvus for reproduction or distribution purposes.

Does Milvus support non-x86 architecture?

No, it does not.

Does Milvus support CRUD operations on vectors?

Yes. To update a vector, you can delete it and then insert a new one.

Can Milvus handle datasets of up to a 100-billion scale?

By deploying Mishards, a cluster sharding middleware for Milvus, you can process datasets of up to a 100-billion scale.

Where does Milvus store imported data?

Vectors imported into Milvus are stored locally at milvus/db/tables/.

Metadata can be stored in either MySQL or SQLite. See Manage Metadata with MySQL for more information.

Why can't I find vectors on SQLite or MySQL?

Milvus stores vectors and indexes directly in the disk as files, not in SQLite or MySQL. It uses SQLite or MySQL to store metadata of the vectors instead.

Can I use SQL Server or PostgreSQL to store metadata in Milvus?

No, we only support storing metadata using SQLite or MySQL.

Does Milvus' Python SDK have a connection pool?

Python SDKs corresponding to Milvus v0.9.0 or later have a connection pool. There is no upper limit on the default number of connections in a connection pool.

Does Milvus support inserting while searching?

Yes.

Is there a graphical tool for managing Milvus?

As of Milvus v0.7.0, we have provided Milvus Enterprise Manager as a graphical tool for managing Milvus.

Can I export data from Milvus?

We do not have a dedicated tool as yet. You can call `get_entity_by_id` to get the intended vectors by ID.

Why do the retrieved vectors suffer precision loss after the `get_entity_by_id` method call?

Milvus stores and processes each dimension of a vector in single-precision floating-point format (accurate to seven decimal places). Therefore, if the original format of each dimension is double-precision floating-point (accurate to sixteen decimal places), you will see a precision loss.

Should I specify entity IDs when importing vectors or have Milvus generate them for me?

Either way is fine. But please note that entity IDs in the same collection must be either user-generated or Milvus-generated. Can't be both.

Can I insert vectors with existing IDs?

Yes, you can. If you insert vectors with an existing ID, you would end up having duplicate IDs.

Is there a length limit on the self-defined entity IDs?

Entity IDs must be non-negative 64-bit integers.

Is there a volume limit on the vectors inserted each time?

Vectors inserted each time must not exceed 256 MB.

Why is the `top1` result of a vector search not the search vector itself, if the metric type is inner product?

This occurs if you have not normalized the vectors when using inner product as the distance metric.

Does the size of a collection affect vector searches in one of its partitions, especially when it holds up to 100 million vectors?

No. If you have specified partitions when conducting a vector search, Milvus searches the specified partitions only.

Does Milvus load the whole collection to the memory if I search only certain partitions in that collection?

No, Milvus only loads the partitions to search.

Are queries in segments processed in parallel?

Yes. But the parallelism processing mechanism varies with Milvus versions.

Suppose a collection has multiple segments, then when a query request comes in:

- CPU-only Milvus processes the segment reading tasks and the segment searching tasks in pipeline.
- On top of the abovementioned pipeline mechanism, GPU-enabled Milvus distributes the segments among the available GPUs.

See How Does Milvus Schedule Query Tasks for more information.

How to choose an index in Milvus?

It depends on your scenario. See How to Choose an Index in Milvus for more information.

Are indexes partition-specific?

No. A collection can have only one index type at a time.

Can Milvus create different types of index in the same collection?

No. Although a collection can hold various types of data, the same collection can use only one index type.

Does Milvus create new indexes after vectors are inserted?

Yes. When the inserted vectors grow to a specified volume, Milvus creates a new segment and starts to create an index file for it at the same time. The building of the new index file does not affect the existing index files.

Does IVF_SQ8 differ from IVF_SQ8H in terms of recall rate?

No, they have the same recall rate for the same dataset.

What is the difference between FLAT index and IVF_FLAT index?

IVF_FLAT index divides a vector space into `nlist` clusters. If you keep the default value of `nlist` as 16384, Milvus compares the distances between the target vector and the centers of all 16384 clusters to get `nprobe` nearest clusters. Then Milvus compares the distances between the target vector and the vectors in the selected clusters to get the nearest vectors. Unlike IVF_FLAT, FLAT directly compares the distances between the target vector and each and every vector.

Therefore, when the total number of vectors approximately equals `nlist`, IVF_FLAT and FLAT has little difference in the way of calculation required and search performance. But as the number of vectors grows to two times, three times, or n times of `nlist`, IVF_FLAT index begins to show increasingly greater advantages.

See How to Choose an Index in Milvus for more information.

Why do I see a surge in memory usage when conducting a vector search immediately after an index is created?

This is because:

- Milvus loads the newly created index file to the memory for the vector search.
- The original vector files used to create the index are not yet released from the memory, because the size of original vector files and the index file has not exceeded the upper limit specified by `cache.cache_size`.

Can I update **`index_file_size`** and **`metric_type`** after creating a collection?

No, you cannot.

What is the interval at which Milvus flushes data to the disk?

Milvus automatically flushes data to disk at intervals of one second.

If I have set **`preload_collection`**, does Milvus service start only after all collections are loaded to the memory?

Yes. If you have set `preload_collection` in server_config.yaml, Milvus' service is not available until it loads all specified collections.

In what way does Milvus flush data?

Milvus loads inserted data to the memory and automatically flushes data from memory to the disk at fixed intervals. You can call `flush` to manually trigger this operation.

What is the recommended configuration for Mishards?

We recommend that you configure write nodes to using GPU-enabled Milvus and read nodes to using CPU-only Milvus. If you can have only one write node, you can configure this node to using GPU-enabled Milvus for creating indexes and configure read nodes to using CPU-only Milvus.

Does Mishards support RESTful APIs?

No, it does not.

What is normalization? Why is normalization needed?

Normalization refers to the process of converting an embedding (vector) so that its norm equals 1. If you use Inner Product to calculate embeddings similarities, you must normalize your embeddings. After normalization, inner product equals cosine similarity.

See Wikipedia for more information.

Why do I get different results using Euclidean distance (L2) and inner product (IP) as the distance metric?

Check if the vectors are normalized. If not, you need to normalize the vectors first. Theoretically speaking, similarities worked out by L2 are different from similarities worked out by IP, if the vectors are not normalized.

Is there a limit on the total number of collections and partitions?

There is no limit on the number of collections. The upper limit on the number of partitions in a collection is 4096.

Why do I get fewer than k vectors when searching for `topk` vectors?

Among the indexes that Milvus supports, IVF_FLAT and IVF_SQ8 implement the k-means clustering method. A data space is divided into `nlist` clusters and the inserted vectors are distributed to these clusters. Milvus then selects the `nprobe` nearest clusters and compares the distances between the target vector and all vectors in the selected clusters to return the final results.

If `nlist` and `k` are large and `nprobe` is small, the amount of vectors in the `nprobe` clusters may be less than `k`. Therefore, when you search for the `topk` nearest vectors, the number of returned vectors is less than k.

To avoid this, try setting `nprobe` larger and `nlist` and `k` smaller.

See Index Types for more information.

What is the maximum dimension of a vector in Milvus?

Milvus can support vectors with up to 32,768 dimensions.

Why does Milvus set an upper limit of 256 MB for the data that can be inserted at a time?

Data inserted to Milvus is first written into memory. This limit is to avoid over-occupation of memory resources.

Milvus merges multi-thread query requests in one batch. How to cancel this mechanism?

You can add the following section to the configuration file server_config.yaml:

```
engine_config:
  search_combine_nq: 1
```

Still have questions?

You can:

- Check out Milvus on GitHub. You're welcome to raise questions, share ideas, and help others.
- Join our Slack community to find more help and have fun!

Operational FAQ

What if I failed to pull Milvus Docker image from Docker Hub?

Users in some countries may have limited access to Docker Hub. In this case, you can pull the Docker image from other registry mirrors. You can add the URL `"https://registry.docker-cn.com"` to the `registry-mirrors` array in `/etc.docker/daemon.json`.

```
{
  "registry-mirrors": ["https://registry.docker-cn.com"]
}
```

Is Docker the only way to install and run Milvus?

No. You can also build Milvus from source code in Linux. See Build Milvus from source code for more information.

Why does Milvus return **`config check error`**?

The version of configuration file does not match the version your Milvus server.

Why do I get **`no space left on device`** when importing data to Milvus?

It is likely that you have not saved enough disk space.

Why is my recall rate unsatisfying?

You can increase the value of `nprobe` when searching from a client. The greater the value, the more accurate the result, and the more time it takes.

See Performance Tuning > Index for more information.

Why does my updated configuration not work?

You need to restart Milvus Docker each time you update the configuration file. See Milvus Server Configuration > Updating configurations.

How can I know if Milvus has started successfully?

Run `sudo docker logs <container ID>` to check if Milvus is running properly.

Why is the time in the log files different from the system time?

The log files in the Docker container use UTC time by default. If your host machine does not use UTC time, then the time in the log files is different. We recommend that you mount the log files onto your host machine to keep the time consistent between the log and the host.

How can I know whether my CPU supports Milvus?

The instruction sets that Milvus supports are SSE42, AVX, AVX2, and AVX512. Your CPU must support at least one of them for Milvus to function properly.

Why does Milvus return **`illegal instruction`** during startup?

If your CPU does not support SSE42, AVX, AVX2, or AVX512, Milvus cannot start properly. You can use `cat /proc/cpuinfo` to check the supported instruction sets.

How can I know whether my GPU is supported by Milvus?

Milvus supports CUDA architecture 6.0 or later. See Wikipedia for supported architectures.

Where is the script for starting the server in the Milvus Docker container?

It is at /var/lib/milvus/script/ in the Milvus Docker container.

Besides the configuration file, how can I tell Milvus is using GPU for search?

Use any of the following methods:

- Use `nvidia-smi` to monitor your GPU usage.
- Use Prometheus to monitor performance metrics. See Visualize Metrics in Grafana > System performance metrics.
- Check the Milvus server logs.

Can I install Milvus on Windows?

Yes, so long as you have set up a Docker environment on your operating system.

How to fix the error when I install pymilvus on Windows?

Try installing pymilvus in a Conda environment.

Can I deploy Milvus service in an air-gapped environment?

Milvus is released as a Docker image. Follow these steps to deploy it from offline:

1. Pull the latest Milvus Docker image when you have Internet access.
2. Run `docker save` to save the Docker image as a TAR file.
3. Transfer the TAR file to the air-gapped environment.
4. Run `docker load` to load the file as a Docker image.

For more information about Docker, see docs.docker.com.

How can I differentiate if I have multiple Milvus nodes connected to Pushgateway?

You can add a Prometheus instance in prometheus.yaml. Then Prometheus or Granafa will show the monitoring data, as well as the source node.

Which database system should I use to manage Metadata, SQLite or MySQL?

We recommend using MySQL to manage Metadata in production environment.

How to calculate required memory based on the size of the dataset?

Different indexes require different memory space. You can use Milvus sizing tool to calculate the required memory for a vector search.

How to migrate data in Milvus?

Copy the entire db directory of the original Milvus service to the new directory. When restarting the Milvus service, map the copied db directory to the db directory of the Milvus service.

Note: Data formats of different versions may not be compatible with each other. The current data format is backward compatible with Milvus v0.7.0.

Can I increase my storage by adding interfaces such as S3 or GlusterFS?

No, you cannot. Milvus does not support this feature for now.

Why do I see `WARN: increase temp memory to avoid cudaMalloc, or decrease query/add size (alloc 307200000 B, highwater 0 B)` in the log file?

You receive this warning if the graphics memory required for a request is larger than the graphics memory allocated beforehand. The warning merely denotes an insufficient graphics memory. Milvus will expand the graphics memory accordingly.

Why does Milvus return `database is locked`?

If you use SQLite to manage Metadata, you receive this error message when write requests occur frequently. We recommend using MySQL for Metadata management. See Manage Metadata with MySQL.

Got an error `Segmentation Fault` from PyMilvus. What shall I do?

PyMilvus v1.1.0 allows you to download the latest version of the grpcio library. The server of Milvus v1.x is built on a earlier, customized version of gRPC, which is incompatible with the latest grpcio library. To fix this issue, either upgrade your PyMilvus version from v1.1.0 to v1.1.1 or roll back your grpcio library to 1.37.0:

```
pip install grpcio==1.37.0
pip install grpcio-tools==1.37.0
```

Can I specify the timeout for `Milvus()`?

Yes, PyMilvus v1.1.2 supports specifying the server connection timeout.

How to assign GPU devices to index/search?

In server_config.yaml file under /home/$USER/milvus/conf, you can assign GPU devices to index/search. See example:

```
gpu:
  enable: true
  cache_size: 10GB
  gpu_search_threshold: 0
  search_devices:
    - gpu0
    - gpu1
  build_index_devices:
    - gpu2
    - gpu3
```

GPU 0,1,2,3 are first 4 GPUs from the list of GPU devices assigned to the docker container.

Still have questions?

You can:

- Check out Milvus on GitHub. You're welcome to raise questions, share ideas, and help others.
- Join our Slack community to find more help and have fun!

Troubleshooting

Troubleshoot

This page describes the common issues you may run into with Milvus. The issues fall into the following categories:

Startup issue

Issues that occur at the startup of Milvus server, and that generally lead to server startup failures. You can check the corresponding error messages by below command:

```
$ docker logs <milvus container id>
```

Operational issue

Issues that occur during the server operation, which may cause server down. If you encounter issues during operation, first confirm that whether the issue is caused by the incompatibility of Milvus and SDK versions. Then check the error messages that are recorded in /home/$USER/milvus/logs.

API issue

Issues that occur during the operation with Milvus through APIs. Corresponding error messages will be returned in real time to the client.

If you cannot resolve the issue easily yourself, you can:

- Join our Slack channel and reach out for support from Milvus team.
- File an Issue on GitHub and describe the problem in detail.

API Reference

Tools

Overview

Milvus Developer Tool Overview

You can use the following tools to improve your developing experience with Milvus.

- Milvus Enterprise Manager: An intuitive GUI tool for managing and interacting with the Milvus server.
- Milvus Sizing Tool: A utility for estimating your hardware configuration based on your application scenario.
- MilvusDM: A data migration tool for importing and exporting Milvus data files.

MilvusDM

Overview

MilvusDM (Milvus Data Migration) is an open-source tool designed specifically for importing and exporting data files with Milvus. MilvusDM can greatly improve data mangement efficiency and reduce DevOps costs in the following ways:

- Faiss to Milvus: Imports unzipped data from Faiss into Milvus.
- HDF5 to Milvus: Imports HDF5 files into Milvus.
- Milvus to Milvus: Migrates data from a source Milvus to the target Milvus.
- Milvus to HDF5: Saves the data in Milvus as HDF5 files.

MilvusDM is hosted on Github and can be easily installed by running the command line `pip3 install` pymilvusdm. MilvusDM allows you to migrate data in a specific collection or partition. In the following sections, we will explain how to use each data migration type.

Faiss to Milvus

Steps

1.Download F2M.yaml:

```
$ wget https://raw.githubusercontent.com/milvus-io/milvus-tools/main/yamls/F2
```

2.Set the following parameters:

- `data_path`: Data path (vectors and their corresponding IDs) in Faiss.

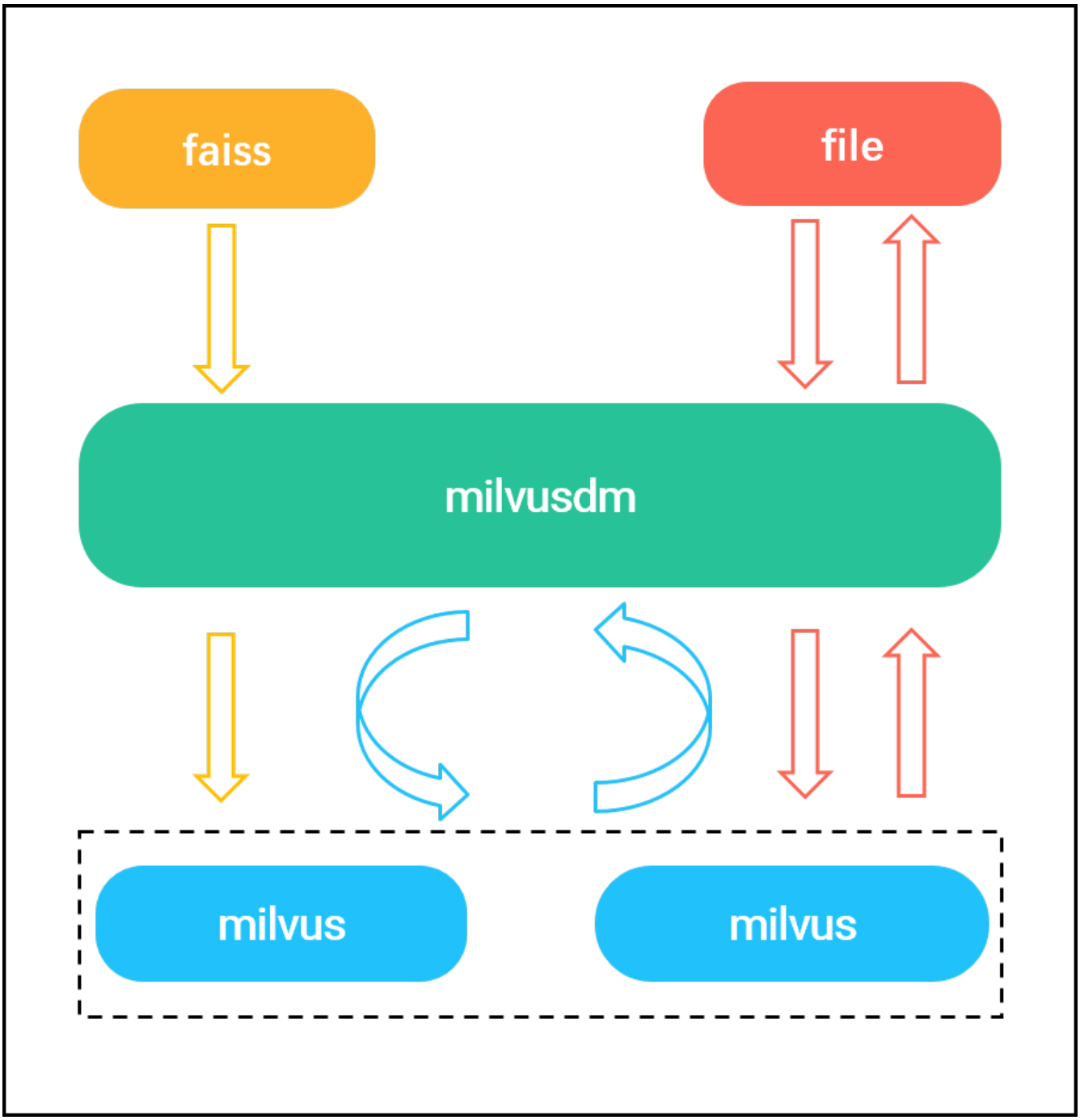

Figure 22: milvusdm blog 1.png

- `dest_host`: Milvus server address.
- `dest_port`: Milvus server port.
- `mode`: Data can be imported to Milvus using the following modes:
  - Skip: Ignore data if the collection or partition already exists.
  - Append: Append data if the collection or partition already exists.
  - Overwrite: Delete data before insertion if the collection or partition already exists.
- `dest_collection_name`: Name of receiving collection for data import.
- `dest_partition_name`: Name of receiving partition for data import.
- `collection_parameter`: Collection-specific information such as vector dimension, index file size, and distance metric.

```
F2M:
  milvus_version: 1.1.0
  data_path: '/home/data/faiss.index'
  dest_host: '127.0.0.1'
  dest_port: 19530
  mode: 'append'        ### 'skip/append/overwrite'
  dest_collection_name: 'test'
  dest_partition_name: ''
  collection_parameter:
    dimension: 256
    index_file_size: 1024
    metric_type: 'L2'
```

3.Run F2M.yaml:

```
$ milvusdm --yaml F2M.yaml
```

Sample Code

1.Read Faiss files to retrieve vectors and their corresponding IDs.

```
ids, vectors = faiss_data.read_faiss_data()
```

2.Insert the retrieved data into Milvus:

```
insert_milvus.insert_data(vectors, self.dest_collection_name, self.collection_parameter, self.mode,
ids, self.dest_partition_name)
```

HDF5 to Milvus

Steps

1.Download H2M.yaml.

```
$ wget https://raw.githubusercontent.com/milvus-io/milvus-tools/main/yamls/H2M.yaml
```

2.Set the following parameters:

- `data_path`: Path to the HDF5 files.
- `data_dir`: Directory holding the HDF5 files.
- `dest_host`: Milvus server address.
- `dest_port`: Milvus server port.
- `mode`: Data can be imported to Milvus using the following modes:
  - Skip: Ignore data if the collection or partition already exists.

  - Append: Append data if the collection or partition already exists.
  - Overwrite: Delete data before insertion if the collection or partition already exists.
- `dest_collection_name`: Name of receiving collection for data import.
- `dest_partition_name`: Name of receiving partition for data import.
- `collection_parameter`: Collection-specific information such as vector dimension, index file size, and distance metric.

  Set either `data_path` or `data_dir`. Do not set both. Use `data_path` to specify multiple file paths, or `data_dir` to specify the directory holding your data file.

```
H2M:
  milvus-version: 1.1.0
  data_path:
    - /Users/zilliz/float_1.h5
    - /Users/zilliz/float_2.h5
  data_dir:
  dest_host: '127.0.0.1'
  dest_port: 19530
  mode: 'overwrite'        ### 'skip/append/overwrite'
  dest_collection_name: 'test_float'
  dest_partition_name: 'partition_1'
  collection_parameter:
    dimension: 128
    index_file_size: 1024
    metric_type: 'L2'
```

3.Run H2M.yaml:

```
$ milvusdm --yaml H2M.yaml
```

Sample Code

1.Read the HDF5 files to retrieve vectors and their corresponding IDs:

```
vectors, ids = self.file.read_hdf5_data()
```

2.Insert the retrieved data into Milvus:

```
ids = insert_milvus.insert_data(vectors, self.c_name, self.c_param, self.mode, ids,self.p_name)
```

Milvus to Milvus

Steps

1.Download M2M.yaml.

```
$ wget https://raw.githubusercontent.com/milvus-io/milvus-tools/main/yamls/M2M.yaml
```

2.Set the following parameters:

- `source_milvus_path`: Source Milvus work path.
- `mysql_parameter`: Source Milvus MySQL settings. If MySQL is not used, set mysql_parameter as ' ' .
- `source_collection`: Names of the collection and its partitions in the source Milvus.
- `dest_host`: Milvus server address.
- `dest_port`: Milvus server port.
- `mode`: Data can be imported to Milvus using the following modes:
  - Skip: Ignore data if the collection or partition already exists.

- Append: Append data if the collection or partition already exists.
- Overwrite: Delete data before insertion if the collection or partition already exists.

```
M2M:
  milvus_version: 1.1.0
  source_milvus_path: '/home/user/milvus'
  mysql_parameter:
    host: '127.0.0.1'
    user: 'root'
    port: 3306
    password: '123456'
    database: 'milvus'
  source_collection:
    test:
      - 'partition_1'
      - 'partition_2'
  dest_host: '127.0.0.1'
  dest_port: 19530
  mode: 'skip' ### 'skip/append/overwrite'
```

3.Run M2M.yaml.

```
$ milvusdm --yaml M2M.yaml
```

Sample Code

1.According to a specified collection or partition's metadata, read the files under milvus/db on your local drive to retrieve vectors and their corresponding IDs from the source Milvus.

```
collection_parameter, _ = milvus_meta.get_collection_info(collection_name)
r_vectors, r_ids, r_rows = milvusdb.read_milvus_file(self.milvus_meta, collection_name, partition_tag)
```

2.Insert the retrieved data into the target Milvus.

```
milvus_insert.insert_data(r_vectors, collection_name, collection_parameter, self.mode, r_ids,
partition_tag)
```

Milvus to HDF5

Steps

1.Download M2H.yaml:

```
$ wget https://raw.githubusercontent.com/milvus-io/milvus-tools/main/yamls/M2H.yaml
```

2.Set the following parameters:

- `source_milvus_path`: Source Milvus work path.
- `mysql_parameter`: Source Milvus MySQL settings. If MySQL is not used, set mysql_parameter as ''.
- `source_collection`: Names of the collection and its partitions in the source Milvus.
- `data_dir`: Directory for holding saved HDF5 files.

```
M2H:
  milvus_version: 1.1.0
  source_milvus_path: '/home/user/milvus'
  mysql_parameter:
    host: '127.0.0.1'
    user: 'root'
    port: 3306
    password: '123456'
```

```
    database: 'milvus'
  source_collection: ### specify the 'partition_1' and 'partition_2' partitions of the 'test' collection.
    test:
      - 'partition_1'
      - 'partition_2'
  data_dir: '/home/user/data'
```

3.Run M2H.yaml:

```
$ milvusdm --yaml M2H.yaml
```

Sample Code

1.According to a specified collection or partition's metadata, read the files under milvus/db on your local drive to retrieve vectors and their corresponding IDs..

```
collection_parameter, version = milvus_meta.get_collection_info(collection_name)
r_vectors, r_ids, r_rows = milvusdb.read_milvus_file(self.milvus_meta, collection_name, partition_tag)
```

2.Save the retrieved data as HDF5 files.

```
data_save.save_yaml(collection_name, partition_tag, collection_parameter, version, save_hdf5_name)
```

MilvusDM File Structure

The flow chart below shows how MilvusDM performs different tasks according to the YAML file it receives:

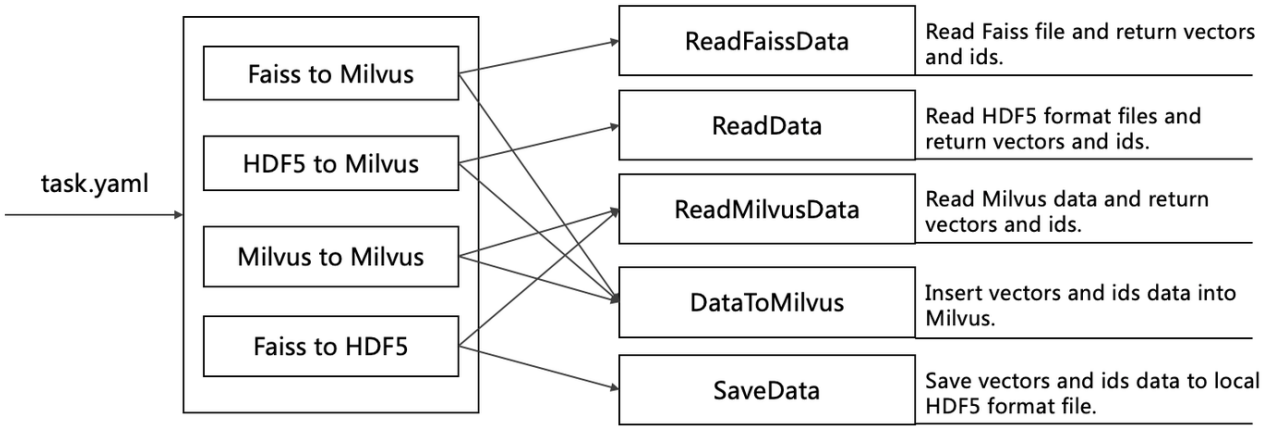

Figure 23: milvusdm blog 2.png

MilvusDM file structure:

- pymilvusdm
  - core
    - milvus_client.py: Performs client operations in Milvus.
    - read_data.py: Reads the HDF5 data files on your local drive. (Add your code here to support reading data files in other formats.)
    - read_faiss_data.py: Reads the data files in Faiss.
    - read_milvus_data.py: Reads the data files in Milvus.
    - read_milvus_meta.py: Reads the metadata in Milvus.
    - data_to_milvus.py: Creates collections or partitions based on parameters in YAML files and imports the vectors and the corresponding vector IDs into Milvus.

        * save_data.py: Saves the data as HDF5 files.
        * write_logs.py: Writes logs during runtime.
    - faiss_to_milvus.py: Imports data from Faiss into Milvus.
    - hdf5_to_milvus.py: Imports data in HDF5 files into Milvus.
    - milvus_to_milvus.py: Migrates data from a source Milvus to the target Milvus.
    - milvus_to_hdf5.py: Exports data in Milvus and saves them as HDF5 files.
    - main.py: Performs corresponding tasks according to the received YAML file.
    - setting.py: Configurations relating to running the MilvusDM code.
- setup.py: Creates pymilvusdm file packages and uploads them to PyPI (Python Package Index).

Recap

MilvusDM primarily handles migrating data in and out of Milvus, which includes Faiss to Milvus, HDF5 to Milvus, Milvus to Milvus, and Milvus to HDF5.

The following features are planned for upcoming releases:

- Import binary data from Faiss to Milvus.
- Blocklist and allowlist for data migration between source Milvus and target Milvus.
- Merge and import data from multiple collections or partitions in source Milvus to a new collection in target Milvus.
- Backup and recovery of Milvus data.

The Milvusdm project is open source and available on Github. Any and all contributions to the project are welcome. Give it a star ☆, and feel free to file an issue or submit your own code!

Release Notes

Release notes

v1.1.0

Release date: 2021-05-07

Compatibility

Milvus version	Python SDK version	Java SDK version	Go SDK version
1.1.0	1.1.0	1.1.1	1.1.0

New Features

- #4564 Supports specifying partition in a `get_entity_by_id()` method call.
- #4806 Supports specifying partition in a `delete_entity_by_id()` method call.
- #4905 Adds the `release_collection()` method, which unloads a specific collection from cache.

Improvements

- #4756 Improves the performance of the `get_entity_by_id()` method call.
- #4856 Upgrades hnswlib to v0.5.0.
- #4958 Improves the performance of IVF index training.

Fixed issues

- #4778 Fails to access vector index in Mishards.
- #4797 The system returns false results after merging search requests with different `topK` parameters.
- #4838 The server does not respond immediately to an index building request on an empty collection.
- #4858 For GPU-enabled Milvus, the system crashes on a search request with a large `topK` (> 2048).
- #4862 A read-only node merges segments during startup.
- #4894 The capacity of a Bloom filter does not equal to the row count of the segment it belongs to.
- #4908 The GPU cache is not cleaned up after a collection is dropped.
- #4933 It takes a long while for the system to build index for a small segment.
- #4952 Fails to set timezone as "UTC + 5:30".
- #5008 The system crashes randomly during continuous, concurrent delete, insert, and search operations.
- #5010 For GPU-enabled Milvus, query fails on IVF_PQ if `nbits` ≠ 8.
- #5050 `get_collection_stats()` returns false index type for segments still in the process of index building.
- #5063 The system crashes when an empty segment is flushed.
- #5078 For GPU-enabled Milvus, the system crashes when creating an IVF index on vectors of 2048, 4096, or 8192 dimensions.

v1.0.0

Release date: 2021-03-09

Compatibility

Milvus version	Python SDK version	Java SDK version	Go SDK version
1.0.0	1.0.x	1.0.x	1.0.x

New Features

- Supports writing log to stdout. #3977

Improvements

- Reduces the package size of grpc-milvus for the C++ SDK. #4754

Fixed issues

- Memory leaks during indexing or querying operations. #4749, #4757, #4765, #4766
    - For more information about the features of Milvus 1.0, go to What's Inside Milvus 1.0?.
    - For more information about its roadmap, see Milvus 1.0: The World's Most Popular Open-Source Vector Database Just Got Better.

Milvus Community Charters

Milvus Community Charters

About the Milvus community

Milvus is an open-source vector database that is highly flexible, reliable, and blazing fast. In June 2021, Milvus graduated from the LF AI & Data foundation's incubator program after spending 15 months in the incubation phase.

We, as the Milvus community, want to build a fundamental data serving component so that AI applications could go production much easier. More and more people start to build up their production AI applications upon Milvus. You can find more user stories of Milvus in our project blog.

If you are a Milvus user or you have the same vision as us, we would warmly welcome you to join our community. Together, we contribute to the project design and participate in the decision making process. Let's build up a community-driven open source AI project.

Join the community

All Contributors could find their positions in the Milvus community. If you are interested in code development, please read thru the chapters of Special Interest Groups and Working Groups. If you want to make non-code contributions, you can find the opportunities in the chapters of Working Groups and Technology Steering Committee.

You could connect with the Milvus community on various channels. Here is the guideline of the community channels.

- Official website: https://milvus.io
- Community calendar for events and meetings
  - https://lists.lfaidata.foundation/g/milvus-announce/ics/7812594/1982604167/feed.ics
- Community announcement
  - Mail list: https://lists.lfaidata.foundation/g/milvus-announce
  - Blog:
- User discussions, Q&A
  - Slack community
  - GitHub discussions
  - Mail list: https://lists.lfaidata.foundation/g/milvus-technical-discuss
- Bug report & feature request
  - GitHub issues
- Group discussions (for special interest groups and working groups)
  - Slack community (1 to 1 conversation, internal group discussion)
  - GitHub discussions (internal group discussion and public community discussion)
  - Mail list: https://lists.lfaidata.foundation/g/milvus-technical-discuss (public community discussion)
- TSC communication
  - Mail list: https://lists.lfaidata.foundation/g/milvus-tsc

Code of Conduct

The Milvus community follows the Milvus Code of Conduct. Meanwhile, as a Linux Foundation project, all Milvus community members should agree to abide by the LF Code of Conduct available at https://lfprojects.org/policies/code-of-conduct/.

Community organization

There are 3 types of groups in the Milvus community.

- Special interest groups (SIGs)
- Working groups (WGs)
- Technology Steering Committee (TSC)

The below diagram shows how they work together.

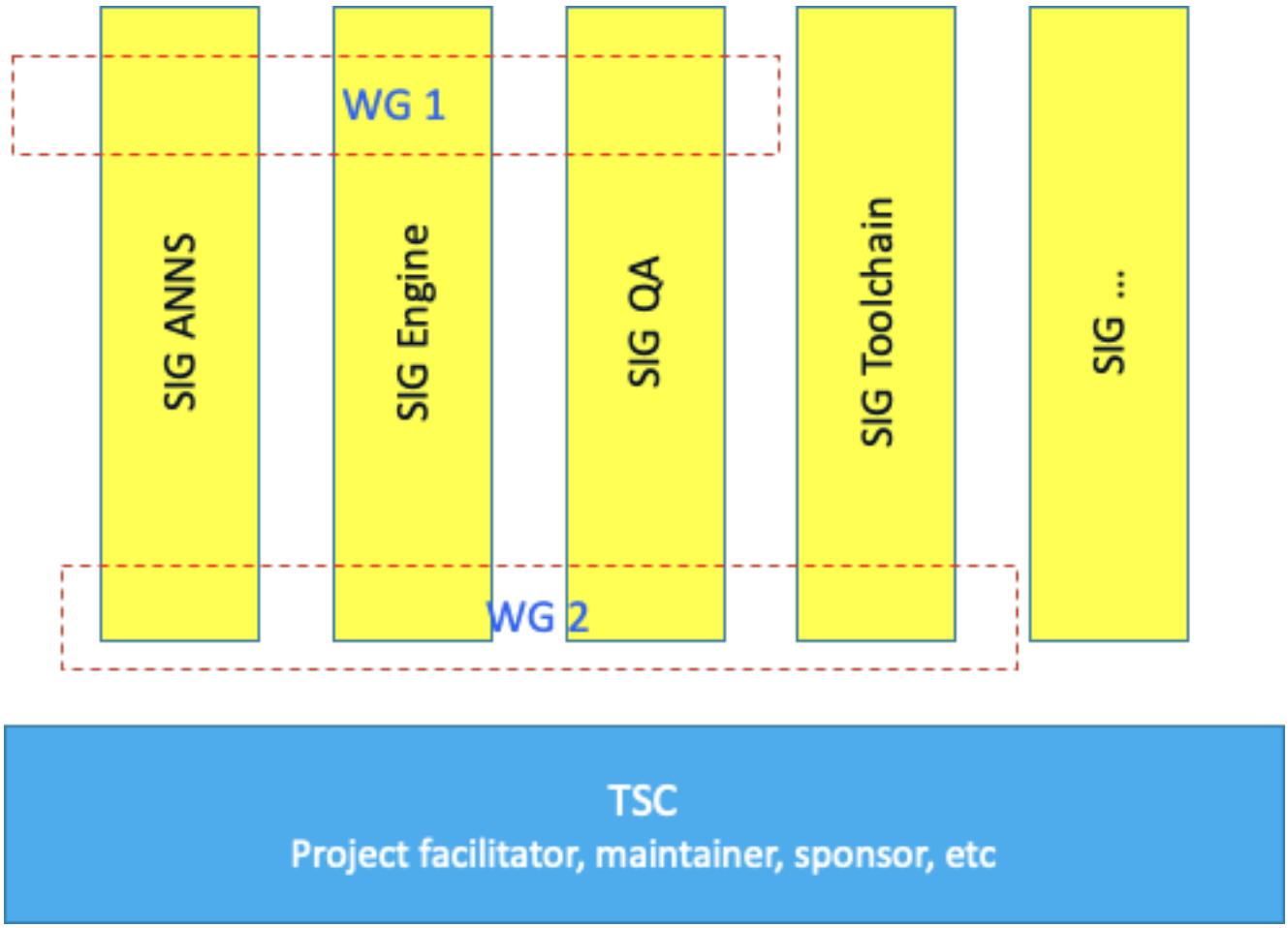

Figure 24: Community Organization

Special Interest Groups (SIGs)

The SIGs are usually aligned to the Milvus system components, technology field, etc. Developers could join the SIGs based on their interests. Below is the list of existing SIGs:

SIGs	Group contact	Group label (GitHub, Slack, etc)
The ANNS algorithm SIG	Yi Xiaomeng, Li Shengjun	sig-anns
The Engine SIG	Yefu Chen, Zhenshan Cao	sig-engine
The Testing SIG	YuDong Cai, Zhenxiang Li	sig-testing
The Toolchain SIG	Shiyu Chen, Yunmei Li	sig-tool

If you want to initiate a new SIG, you need to complete and submit the SIG proposal. The proposal will be reviewed and voted on by Milvus Technology Steering Committee (TSC).

Code contribution guideline

Please refer: Contributing to Milvus

Developer path in SIGs

Participant Once you connect to a SIG (subscribe to the community calendar, read thru the group discussion, attend group meetings, etc), you are already a participant.

Contributor If your code contribution (thru pull request) is successfully merged into the Milvus codebase, you are a contributor.

Reviewer After you have provided continued and quality contribution to the Milvus project for at least 6 months and have contributed at least one major component. You are eligible to become the reviewer of that component. The SIG leader will review and nominate reviewers periodically. You can nominate yourself as well if you want to show more commitment to the Milvus project.

Reviewers have the following responsibilities:

- Participate in feature design discussion and implementation.
- Ensure the quality of owned code modules.
- Ensure the technical accuracy of documentation.
- Quickly respond to issues and PRs and conduct code reviews.

Committer To become a committer, a reviewer must have contributed broadly throughout the Milvus project. A reviewer must also be sponsored by a committer and the sponsorship must be approved by the TSC.

The committer role enables the contributor to commit code directly to the repository, but also comes with the responsibilities of being a leader in the community:

- Lead feature design discussions and implementation.
- Ensure the overall project quality and approve PRs.
- Participate in product release, feature planning, and roadmap design.
- Have a constructive and friendly attitude in all community interactions.
- Mentor reviewers and contributors.
- In general, continue to be willing to spend at least 25% of one' s time working on the project (~1.25 business days per week).

Conflict resolution and voting

In general, we prefer that technical issues and committer membership are amicably worked out between the persons involved. If a dispute cannot be decided independently, the committers can be called in to decide an issue. If the committers themselves cannot decide an issue, the issue will be resolved by voting. The voting process is a simple majority in which each committer receives one vote.

Working Groups (WGs)

Some types of work (for example, release, documentation, etc) requires collaboration across different SIGs. We set up working groups to ensure the smooth proceeding of these types of work. Base on the nature of the work, there are 2 types of working groups, permanent working groups, and on-demand working groups. Below is the list of existing WGs.

Permanent WGs

left	center	right
The Release WG	Xiangyu Wang	wg-release
The Documentation WG	Keith Cai	wg-doc
The DevRel WG	Kate Shao	wg-devrel

On-demand WGs

left	center	right
The Graduation WG	Jun Gu	wg-grad

Technology Steering Committee (TSC)

The Milvus project is legally owned by the Linux Foundation. Milvus TSC is the facilitator of the Milvus project. The TSC members ensure the success of the Milvus project.

Responsibilities:

1. Create and dismiss SIGs and working groups
2. Nominate and vote on new project committers
3. Provide support, guidance, resources, etc. to SIGs and working groups
4. Other Milvus related operation work

TSC Members:

1. Committers
2. Project facilitators (sponsors, project operation leader, invited advisors)

Credits

The contents of this document are based on the Meritocratic Governance Model by Ross Gardler and Gabriel Hanganu with some additions from the TiDB Community Governance.

Special Interest Groups (SIGs)

The ANNS SIG

The ANNS Special Interest Group

About the ANNS Special Interest Group

The group subject

The ANNS special interest group focuses on:

- Research and support more index and metric types in Milvus
- Improve the building and searching efficiency
- Develop autonomous tools for index type and parameter choosing

The group members

- Group leaders
  - Xiaomeng Yi (GitHub: yxm1536)
  - Shengjun Li (GitHub: shengjun1985)
- Group members
  - QianYa Chen (GitHub: cqy123456)
  - ChengMing Li (GitHub: op-hunter)

The group communication channels

- Slack community (1 to 1 conversation, internal group discussion)
- GitHub discussions (internal group discussion and public community discussion)
- Mail list: https://lists.lfaidata.foundation/g/milvus-technical-discuss (public community discussion)

The Engine SIG

The Engine Special Interest Group

About the Engine Special Interest Group

The group subject

The Engine special interest group focuses on:

- Adapt Milvus engine into the cloud-native environment
- Switch to golang in the future development of the Milvus engine (while the ANNS algorithm component will still be developed by C++)

The group members

- Group leaders
  - Yefu Chen (GitHub: neza2017)
  - Zhenshan Cao (GitHub: czs007)

The group communication channels

- Slack community (1 to 1 conversation, internal group discussion)
- GitHub discussions (internal group discussion and public community discussion)
- Mail list: https://lists.lfaidata.foundation/g/milvus-technical-discuss (public community discussion)

The Testing SIG

The Testing Special Interest Group

About the Testing Special Interest Group

The group subject

The Testing special interest group focuses on:

- Build the efficient testing service and reliable、continuous integration facilities, to make it easier and more efficient for the community to run test cases
- Working with other sig groups, find new issues in the project, and close fixed issues relying on the stable testing infrastructure and services

The subgroups in Testing SIG

- test-infra

Build CI/CD infrastructure and pipelines for the project, publish friendly deployment for the project to make it easier to create, install and upgrade the environment

- test-framework

Build layered testing scripts, tools, and services, schedule and execute the test tasks (including functional test, benchmark test, and stability test)

- test-commons

Focus on the ease of use around the server and SDKs, including the execution of functional testing, to ensure the server and SDKs are published and delivered under the original demands and expected quality

The group members

- Group leaders
  - Yudong Cai (GitHub: cydrain)
  - Zhenxiang Li (GitHub: del-zhenwu)
- Group members
  - Zhifeng Zhang (GitHub: jeffoverflow)
  - Yufen Zong (GitHub: ThreadDao)
  - Ting Wang (GitHub: wangting0128)

The group communication channels

- Slack community (1 to 1 conversation, internal group discussion)
- GitHub discussions (internal group discussion and public community discussion)
- Mail list: https://lists.lfaidata.foundation/g/milvus-technical-discuss (public community discussion)

The Toolchain SIG

The Toolchain Special Interest Group

About the Toolchain Special Interest Group

The group subject

The Toolchain special interest group focuses on:

- Good data management. Makes it easier to manage Milvus data.
- Other tools that reduce the effort for Milvus administration and maintenance.
- If you have any good ideas for Milvus Toolchain, let’ s discuss here.

The group members

- Group leaders
  - Chen Shiyu (GitHub: shiyu22)
  - Li Yunmei (GitHub: Bennu-Li)
- Group members
  - Li Qing (GitHub: miia12)
  - Jia Jingjing (GitHub: jingkl)

The group communication channels

- Slack community (1 to 1 conversation, internal group discussion)
- GitHub discussions (internal group discussion and public community discussion)
- Mail list: https://lists.lfaidata.foundation/g/milvus-technical-discuss (public community discussion)

Working Groups (WGs)

The Release WG

The Release Working Group

About the Release Working Group

The group subject

The release working group (WG-release) ensures the quality of the Milvus release:

- Set up and maintain the test automation infrastructure for CI/CD, integration testing, benchmark testing, etc
- Build the regular release in a timely manner
- Coordinate with different development SIGs for a new milestone release

The group members

- Group leaders
  - Xiangyu Wang (GitHub: scsven)

The group communication channels

- Slack community (1 to 1 conversation, internal group discussion)
- GitHub discussions (internal group discussion and public community discussion)
- Mail list: https://lists.lfaidata.foundation/g/milvus-technical-discuss (public community discussion)

Special communication rules: - The release note of any release should be posted to ， milvus-announce@lists.lfaidata.foundation

Reference

- The Milvus Release Guideline

The DevRel WG

The DevRel Working Group

About the DevRel Working Group

The group subject

The DevRel working group focuses on: - Community Outreach

```
-Speaking at conferences/giving talks.

-Being active on social media and technical forums to promote Milvus.

-Creating content such as blogs, video tutorials, newsletters, etc.

-Hosting/attending events (meetups, webinars, hackathons, workshops, DevRel working group’s regular meeti
```

- Developer Experience & Success

  -Facilitating smooth developer onboarding by creating documentation and tools.

  -Building demos.

  -Answering questions or providing guidance to developers/users on GitHub, StackOverflow, Slack, etc.

  -Understanding user pain points and giving feedback to the product team for creating better products.

- Community Building

  -Continuously developing and refining Milvus Ambassador Program to keep the DevRel working group growing.

  -Helping community members to find and join other appropriate Milvus SIGs.

The group members

- Group leaders
  - Kate Shao (GitHub: kateshaowanjou)

The group communication channels

- Slack community (1 to 1 conversation, internal group discussion)
- GitHub discussions (internal group discussion and public community discussion)
- Mail list: https://lists.lfaidata.foundation/g/milvus-technical-discuss (public community discussion)

The Graduation WG

The Graduation Working Group

About the Graduation Working Group

The group subject

This is an on-demand working group. The Graduation working group is set up to ensure the successful graduation of Milvus from the incubating stage in the LF AI & Data foundation. The graduation working group will be dismissed afterward.

The objectives:

- Fulfill all the graduation criteria requested by the LF AI & Data foundation
- Prepare and present the Milvus graduation deck to the governing board of the LF AI & Data foundation
- Graduate before Jun. 30th, 2021

The group members

- Group leaders
  - Jun Gu (GitHub: gujun720)

The group communication channels

- Slack community (1 to 1 conversation, internal group discussion)
- GitHub discussions (internal group discussion and public community discussion)
- Mail list: https://lists.lfaidata.foundation/g/milvus-technical-discuss (public community discussion)

Project Guidelines

The Milvus Release Guideline

The Milvus release guideline

Regular release

There are 2 types of regular releases, monthly and weekly.

Monthly feature release

The target date of the monthly feature release is on the 1st Fri. of a month. We will merge the new features into the monthly feature release.

Weekly bugfix release

The target date of the weekly bugfix release is on every Fri. other than the 1st Fri. of a month. We will merge the latest bugfixes into the weekly bugfix release.

Milestone release

When Milvus evolves to a certain stage (milestone), Milvus committers could propose to build a milestone release. The proposal will be voted in Milvus TSC (simple majority).

The initiator should consider the below aspects in the milestone release proposal.

- What is the purpose of the milestone release?
- Is it a long-term support release? What is the end of support (EOS) date?
- What is the future development strategy for this milestone release/branch?

Milvus version number explanation

The traditional software version number follows the format as "version.release.modification". While open source software (OSS) usually builds more frequent, periodical releases, so the style of OSS version number would be quite different.

When Milvus first launched on GitHub, we set the Milvus version number as 0.y.z. The leading "0" means this is a young, early project. The middle "y" increments every time we build a release. And the ending "z" is the fix-pack number of a release.

After we elevate Milvus 0.10.x to Milvus 1.0 beta. The format of the Milvus version number would be x.y.z.

- x: current milestone + 1.

- y: starting from 0, incrementing after every monthly regular release.
- z: starting from 0, incrementing after every weekly regular release, recycled after next monthly regular release.

For example, after we release Milvus 1.0.0 milestone, the developing version would start from 2.0.0.