zilliz

Milvus Performance Evaluation 2023

TECHNICAL PAPER

Release Date

February 10th, 2023

Authors

Rentong Guo, Li Liu, Chun Han, Ting Wang, Yuchen Gao, Jie Zeng, Chao Gao, Filip Haltmayer, Xiaofan Luan

Table of Contents

Executive Summary

On January 25th, 2022, Milvus 2.0.0 was released. Over the last year, we have made a lot of progress thanks to the joint effort of the entire Milvus community.

- **12 Releases** - Milvus launched 12 releases in the past year.
- **22 Trendings** - The Milvus project was listed on the GitHub Trending Repositories 22 times, staying on the list for a total of 45 days.
- **16K+ Commits** - Through the hard work of all 233 Milvus contributors, a total of 16347 commits were made.

Thanks to all these contributions, Milvus rose to a new level, and how better to show it than to provide a performance report. In this report, the engineering team at Zilliz compared each key release since Milvus 2.0.0, comparing the search latencies and throughput across four well known datasets (DEEP, GIST, GloVe, SIFT) from ann-benchmarks. In addition to this, this report also evaluates the scalability of Milvus 2.2.3 at the billion scale, using both the BIGANN and LAION datasets.

To the best of our knowledge, Milvus is the first vector database proving its capability of serving billion-scale datasets in a public report.

I. Key Findings

2.5x

2.5x reduction in search latency

With Milvus 2.2.3 we achieved a 2.5x reduction in search latency compared to the original Milvus 2.0.0 release. Having a lower latency is beneficial for all users, but is key when dealing with real-time information retrieval systems such as recommendation systems, image/video/text search, and question answering.

4.5x

4.5x increase in throughput

Through our testing under identical environment, we saw a 4.5x increase in QPS with Milvus 2.2.3 compared to 2.0.0. Due to this increase, Milvus is now even more hardware and cost efficient, which is key in building large-scale vector search platforms.

Billion-scale

Scale-out for billion-scale collection with little performance degradation

As we scaled-out a Milvus 2.2.3 cluster for a growing collection, little performance degradation was seen in both search latency and QPS. The performance stayed stable even if we scaled the collection size from million-scale to billion-scale.

Multi-replica

Linear scalability on multi-replicas

Milvus 2.2.3 also showed linear scalability when using multiple replicas. For applications with high throughput requirements, QPS can be easily scaled up and down by simply adding or removing replicas, with no cost-efficiency degradation.

Overall, this paper aims to provide an up-to-date evaluation of Milvus's current state. After each key release we will be updating the experimental results with our new findings. All benchmarking code is open sourced and available at zilliztech/vectordb-benchmark. As always, feel free to open an issue for any questions or suggestions you may have.

Environment

I. About Milvus

Milvus is an open-source vector database built to power embedding similarity search and AI applications. Milvus makes unstructured data search more accessible, and provides a consistent user experience regardless of the deployment environment. Milvus 2.0 is a cloud-native vector database with storage and computation separated by design. All components in this refactored version of Milvus are stateless to enhance elasticity and flexibility. For more architecture details, see Milvus Architecture Overview.

The versions we used in our evaluations are Milvus 2.0.0, Milvus 2.2.0, and Milvus 2.2.3. The table below gives a brief summary of these versions.

Table 1. The Milvus versions used in the evaluations

Milvus	Release Date	Release Notes
Milvus 2.0.0	25 January, 2022	https://milvus.io/docs/v2.0.x/release_notes.md#v200
Milvus 2.2.0	18 November, 2022	https://milvus.io/docs/release_notes.md#v220
Milvus 2.2.3	10 February, 2023	https://milvus.io/docs/release_notes.md#223

For the detailed Milvus configurations, see configures for standalone deployment and configures for cluster deployment.

2. The Datasets

The benchmark consists of six datasets. DEEP, GIST, GloVe, and SIFT are the four datasets from ann-benchmarks that are used to evaluate the search performance of the different Milvus versions. Billion-scale ANNS benchmark's BIGANN and LAION's LAION-5B datasets are the two billion-scale datasets used to evaluate the scalability of Milvus 2.2.3. Table 2 gives a summary of the datasets.

Table 2. The datasets

Dataset	Dimensions	Data type	Distance	Size
DEEP	96	float32	IP	9,990,000
GIST	960	float32	L2	1,000,000
GloVe	100	float32	IP	1,183,514
SIFT	128	float32	L2	1,000,000
BIGANN	128	uint8	L2	1,000,000,000
LAION	768	float32	L2	1,000,000,000

3. Hardware and Deployment

For the evaluations done in this paper, Milvus instances are deployed on GCP general purpose n2-standard-8 or n2-highmem-16 machines. When in standalone, Milvus runs on n2-standard-8. When in cluster mode, the cluster is deployed on multiple n2-highmem-16 machines with Milvus components spread across them.

All the Milvus instances are deployed via the Milvus Operator. This is a solution that deploys and manages a full Milvus service stack on Kubernetes (K8s) clusters.

Table 3. The n2-standard-8 and n2-highmem-16 instance types

Machine type	vCPUs	Memory (GB)	Local SSD (GB)	Default egress bandwidth (Gbps)
n2-standard-8	8	32	100	16
n2-highmem-16	16	128	200	32

Methodology and Evaluation Results

1. ANN Search Latency

The purpose of this test is to compare the search latency of Milvus 2.0.0, Milvus 2.2.0, and Milvus 2.2.3.

The datasets used in this test (DEEP, GIST, GloVe, SIFT) each contain roughly one million vectors. From each dataset we select 10,000 query vectors and for each of these query vectors we use their top 100 closest vectors as ground truths. The same exact data is used across all tests.

For each test we deploy a Milvus instance in standalone mode, load the data in, and index it. Once indexed, we begin our searches with each search request containing one query vector. We run each of our experiments for 60 seconds, with an additional warmup time of 30 seconds.

Due to the impacts of search parameters on search latency and accuracy, we decided to configure the parameters to control the recall at 0.95, the commonly adopted recall rate in many real world applications.

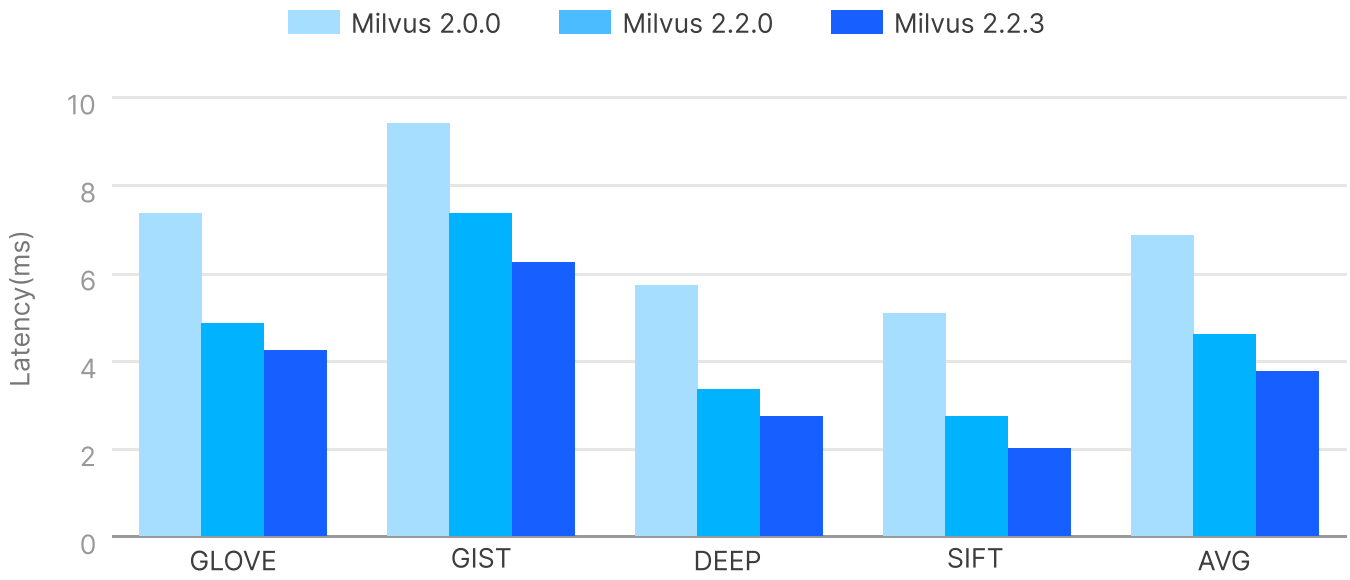

Figure 1. Search latency of Milvus 2.0.0, 2.2.0, and 2.2.3 on GLOVE, GIST, DEEP, SIFT

As seen in Figure 1, all the tested cases have latencies below 10ms, indicating that all the key releases post 2.0.0 are suitable for realtime information retrieval applications. In addition to this, we observed consistent performance improvements through each key release, with Milvus 2.2.0 and 2.2.3 achieving an average speedup of 1.5x and 1.8x respectively compared to 2.0.0. On SIFT, Milvus 2.2.3 demonstrated a 2.5x maximum speedup, reducing the search latency from 5.2 ms (Milvus 2.0.0) to 2.1 ms.

2. ANN Search Throughput

The purpose of this test is to compare the difference in QPS between Milvus 2.0.0, Milvus 2.2.0, and Milvus 2.2.3. The design of the throughput test is similar to the latency test, except that we set the client number to 100 to increase concurrency.

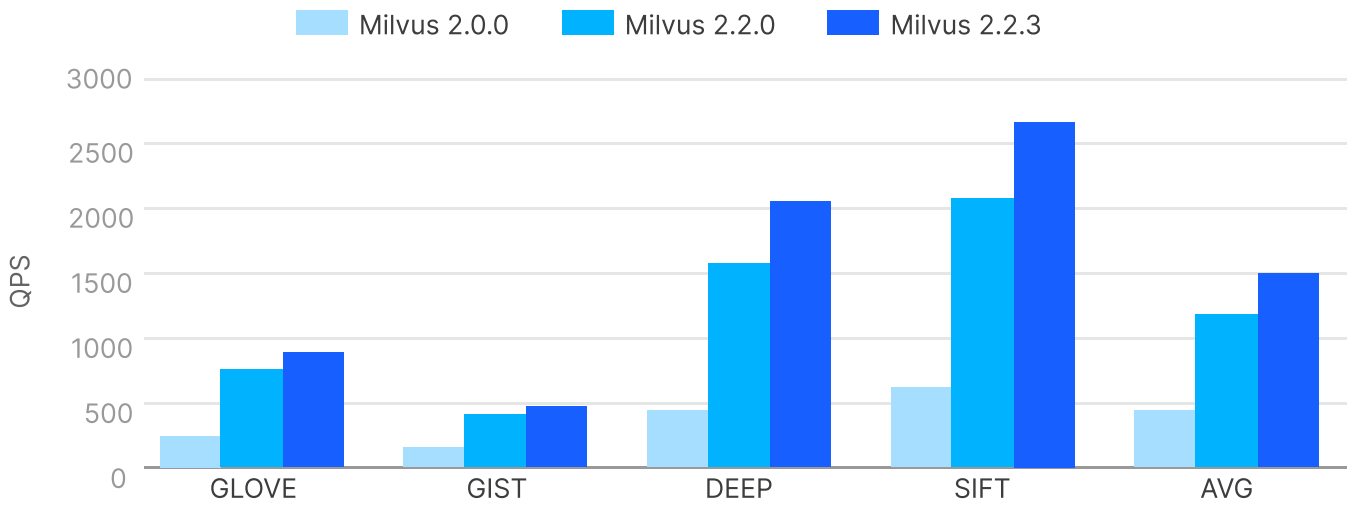

Figure 2. QPS of Milvus 2.0.0, 2.2.0, and 2.2.3 on GLOVE, GIST, DEEP, SIFT

According to the evaluation results seen in Figure 2, the improvements in throughput are consistent with the releases, just like we saw with the latency improvements. The throughput improvements from Milvus 2.0.0 to Milvus 2.2.0 are significant with a 3.4x higher QPS on DEEP and SIFT and a total average increase of 3.1x. In addition to this, Milvus 2.2.3 brings even greater improvements, showing up to 4.5x higher QPS on DEEP and SIFT, equating to 2K+ QPS from a single instance. Overall the average improvement of 2.2.3 compared to 2.0.0 is 3.8x.

There are many optimizations that contribute to these throughput (as well as latency) improvements. Since the core of a vector database is high-dimensional space calculations, the curse of dimensionality exists. To play against the explosive nature of increased data dimension, sparsity, and access randomness, Milvus introduced a set of optimization techniques. These techniques include but are not limited to indexing and searching algorithms, memory management, copy reduction, data layout optimization, multi-threading, and SIMD acceleration.

3. Latency vs. Throughput

Latency and throughput are two of the key metrics when looking at search performance. Ideally, we want low latency along with high throughput. However, these two metrics are related when we take hardware resources and request concurrency into consideration. Figure 3 illustrates the latency and QPS on GloVe (the trend remains the same on other datasets). In the case of low request concurrency (1 client thread), there are enough resources to serve each request upon arrival. The request queue is nearly empty and there is no wait time, resulting in a very low search latency. Although this is good for latency, there are not enough requests to keep the system busy, resulting in a low QPS. In contrast, in the case of high request concurrency (8 client thread), the length of the request queue will grow and end up holding enough requests to avoid system idling. As a result, QPS will increase at the cost of latency increasing due to the added queuing time.

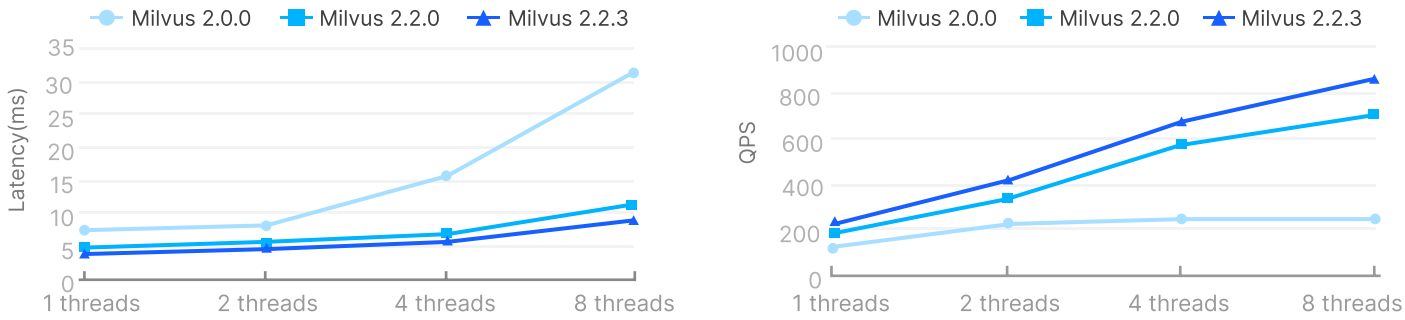

Figure 3. Latency and QPS of Milvus 2.0.0, Milvus 2.2.0, and Milvus 2.2.3 on GloVe under different request concurrency.

Note that Milvus 2.2.3 always outperforms Milvus 2.0.0 and Milvus 2.2.0, no matter how high the concurrency, with it always having the lowest latency curve along with the highest QPS curve. In addition to this, Milvus 2.2.3 shows a lower latency increase rate (lighter slope) as the concurrency doubles. This is the due to end-to-end optimization in search request processing, including optimizations in scheduling, searching, and reducing. Milvus 2.2.3 also shows a higher QPS increase rate (steeper slope), which implies more efficient hardware resource utilizations.

4. Scalability at the Billion-scale Dataset

In many real-world scenarios, data size continues to grow as the business grows, often reaching the billion-scale. Due to this, the cluster needs to be expanded to keep up with the growing data. The key question is, can Milvus keep up in performance as the data size and cluster size continue to grow?

To answer this question, we evaluate how the search latency and QPS varies as we scale the data up to one billion. The datasets used in this test are generated from the LAION dataset. In each round of the experiment, we randomly pick a subset of vectors with the initial dataset containing 65 million vectors. To serve these vectors, we deploy a cluster with three nodes (n2-highmem-16 machines, 16 vCPUs and 128 GB memory, as specified in Table 3), with each node holding around 22 million vectors. In the following rounds of the experiments, we double the data size as well as the cluster size, until the data size reaches one billion.

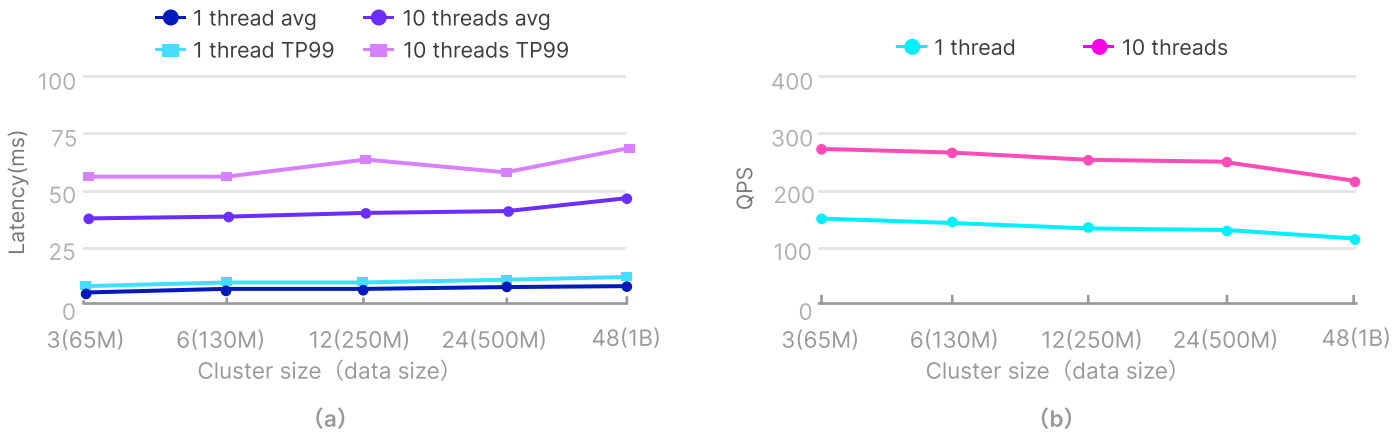

Figure 4. Latency and QPS of different cluster sizes.

Figure 4 illustrates the results. In the case of 1 client thread shown in Figure 4(a), the average latency varies from 6 ms to 8 ms. The low latency variation implies that, as we double the data size and the cluster size, the search latency stays stable, even if the data size is scaled to one billion. In addition to this, the variation of TP99 latency is also low (search latencies fall between 9 ms and 11 ms). This means that the larger cluster size does not introduce more latency spikes. The trends shown in the 10 threads case show similar results, with average and TP99 latencies of 40±5 ms and 62±6 ms respectively. Note that even though the higher request concurrency (10 thread vs. 1 threads) results in larger queuing times, it introduces little impact to the latency variation as we increase cluster size.

Figure 4(b) shows the throughput performance. The QPS is nearly stable as we increase the cluster size. In the 1 thread case, the QPS falls between 115 and 150. In the 10 threads case, the QPS varies between 216 and 273. As we can see, although not significant, there is still some reduction in throughput as we increase cluster size. Many factors can contribute to this result, including network throughput, proxy workload, grpc overhead, scheduling policy, etc. In our studies, we found that the main reason for this comes from the imbalance in data segment distribution across nodes. For the same collection, if we only take data segment size into consideration, larger segments will offer higher search performance, at the cost of longer index building time. However, if we try to map the segments into a cluster, larger segment sizes will harm the balance in data distribution. Milvus adopts a round-robin placement policy for segment-node assignment, but it does not try to divide the remainder (the tail) into smaller segments, because the inefficiency of the small segments will bring even worse performance.

5. Scalability on Multi-replica

Replicas provide redundancy in data by creating copies across the query nodes, protecting against hardware failure and increasing search request throughput. In order to show the improved scalability of Milvus with replication, we tested search performance as the number of replicas increased. The dataset used is derived from the LAION dataset, containing 70 million randomly picked vectors. Each replica contains a complete copy of the dataset, and we add three nodes per replica to the Milvus cluster. In order to provide enough request concurrency, we set the client thread number to 40.

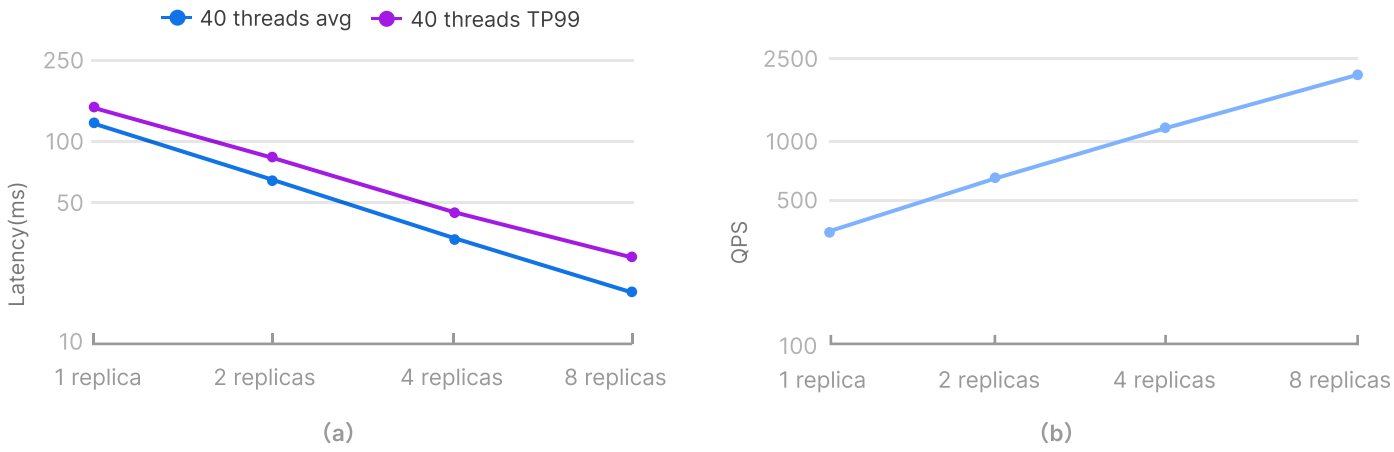

Figure 5. Latency and QPS as replica number doubles.

Figure 5 shows the results, where both axes use logarithmics scales. The results in figure 5(b) show that a single replica is able to achieve approximately 300 QPS while eight replicas are able to reach 2300 QPS. As the number of replicas increases, the system throughput scales linearly. Figure 5(a) shows the results in respect to search latency. In the case of one replica, the average latency and TP99 latency are 123 ms and 143 ms respectively. The largest culprit to this latency is the queuing time, and as we add more replicas, the search latency drops significantly. When the cluster has eight replicas, the average and TP99 latency are decreased to 18 ms and 26 ms respectively. Note that in our experiment, the search latency drops at a near linear rate as the number of replicas increases. But this trend doesn't continue forever, as when the queuing time is reduced to zero, the search latency will remain constant, as the computation time dominates the total time consumed.

Acknowlegement

These great Milvus improvements would not have been possible without the contributions of all the members of Milvus community. We would also like to thank Jiazheng Lu, for his efforts on typography and cover design. We also appreciate the help from Zhenshan Cao, Congqi Xia, and Weida Zhu, for their advice on experiment design and data analysis.

Contact Us

To follow the latest updates on Milvus, or if you have any questions about this evaluation report, please feel free to contact us via:

Milvus

- https://milvus.io/
- https://github.com/milvus-io/milvus
- milvusio.slack.com
- @milvusio
- https://www.youtube.com/c/MilvusVectorDatabase
- https://www.linkedin.com/company/the-milvus-project/

Zilliz

- info@zilliz.com
- https://zilliz.com/
- @zilliz_universe
- https://www.linkedin.com/company/zilliz/