Understanding Scalability Limitations in Distributed Vector Databases

Soobin Cho
Sungshin Women's University
Seoul, Republic of Korea
20231163@sungshin.ac.kr

Gyuyeong Kim
Sungshin Women's University
Seoul, Republic of Korea
gykim@sungshin.ac.kr

ABSTRACT

Vector databases are essential components of modern AI infrastructures, particularly in supporting retrieval-augmented generation (RAG) for large language model (LLM) services. Scaling these databases in distributed and networked environments remains a significant challenge in meeting increasing service demands. However, existing distributed vector database architectures support only capacity scalability, not performance scalability due to query broadcasting and result aggregation overheads. To address these limitations, this paper discusses future research directions aimed at enabling scalable performance in distributed vector databases.

1 SCALABILITY IN VECTOR DATABASES

Vector databases [4] have become essential infrastructure for managing high-dimensional vector data in AI applications, such as large language models (LLMs) with retrieval-augmented generation (RAG). As unstructured data continues to grow rapidly and machine learning (ML) advances, these databases store data features as vector embeddings, enabling efficient similarity search over large-scale datasets.

Scaling out vector databases to networked and distributed environments is a key challenge. In particular, vector database systems are expected to scale in both capacity and performance. However, existing systems such as Milvus[2] primarily focus on capacity scalability. They distribute vector data across multiple nodes to store many vector data, but do not support performance scalability.

For example, Figure 1 illustrates how Milvus handles similarity search in a distributed setting. When a search query arrives, the proxy node broadcasts it to all worker nodes that manage relevant shards. Each worker independently retrieves the necessary vector data from shared storage, performs a local top-k similarity search, and returns its partial results to the proxy. The proxy then merges these partial results to produce the final top-k response for the client.

Despite employing distributed architectures, these systems suffer from performance bottlenecks due to query broadcasting and result aggregation. First, since every node should handle the same query, the throughput improvement with extra nodes is limited. Second, the query latency is determined by the slowest worker node due to the shared-or-nothing nature of the result aggregation. In this context, we ask the following question: *how can we achieve performance scalability in distributed vector databases?*

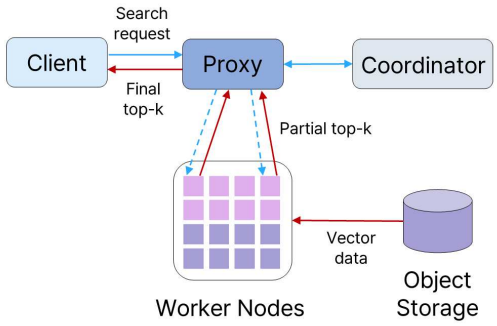

Figure 1: Example of distributed vector search in Milvus [2] ***The proxy node broadcasts search queries to the worker nodes and aggregates partial results, limiting the performance of distributed vector databases.***

2 RESEARCH DIRECTIONS

To achieve scalable performance, it is crucial to both minimize the number of shards each query accesses and accelerate retrieval within each shard. This requires similarity-aware sharding in conjunction with a global index that spans all vectors. Such an approach allows for efficient query routing and localized nearest neighbor search, thereby improving overall system scalability.

Vector indexing primer. There are three widely adopted indexing techniques: Inverted File (IVF), Hierarchical Navigable Small World (HNSW), and Inverted File Product Quantization (IVF_PQ). Figure 2 illustrates how queries are processed with IVF, HNSW, and IVF_PQ. IVF partitions the vector space into clusters by using clustering algorithms (e.g., K-means), and the system scans only the vectors in a few relevant clusters, effectively reducing the search space and improving query speed. HNSW constructs a hierarchical graph connecting vectors to their nearest neighbors, enabling fast

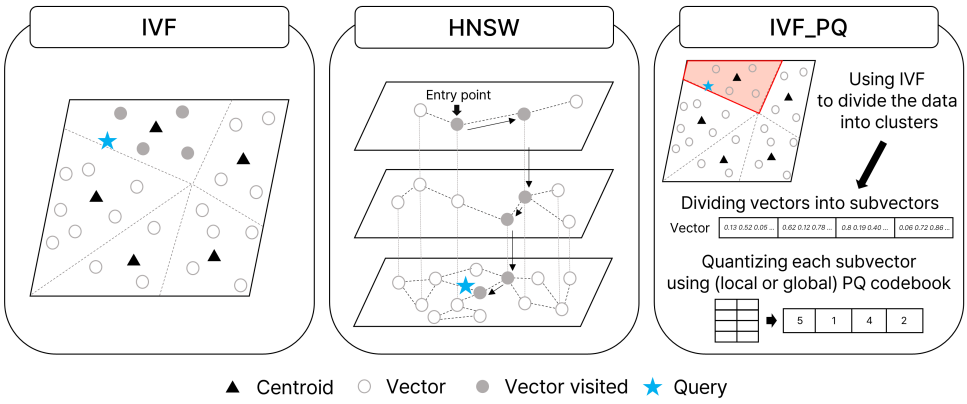

Figure 2: Query processes of approximate nearest neighbor search using IVF, HNSW, and IVF_PQ

navigation through sparse upper layers and precise refinement in denser lower layers. IVF_PQ combines IVF with Product Quantization (PQ) to enable efficient similarity search in high-dimensional vector spaces. It significantly narrows the search space and compresses high-dimensional vectors into smaller and more efficient representations using pre-trained codebooks.

Distributed, global vector indexing. While existing indexing techniques are originally designed to improve performance within a single server, they can be extended to distributed environments for building global indexes. First, to minimize the number of nodes that a request should visit, we may construct a global index by compressing vector data and IVF centroids, which are originally sharded across multiple shards. Each vector associated with a centroid can be divided and compressed using PQ, achieving compact storage and enabling efficient approximate vector similarity search by determining a set of candidate destination nodes where a query should visit. This selective querying may reduce the system loads, improving throughput.

Early termination for fast result aggregation. To ensure bounded query latency, early termination methods [3] can be employed. Instead of processing all candidate vectors, the system monitors intermediate results and stops the search once a predefined accuracy or error threshold is met. Integrating early termination with indexing and quantization-based filtering can further eliminate unnecessary computation, significantly reducing query latency.

More scalability with eBPF. Combining our design direction into eBPF [1], an emerging high-performance host packet processing technique, may further contribute to achieving high-performance vector search in distributed environments. Specifically, we may improve the performance of the proxy node that broadcasts queries and aggregates partial results. Currently, the user-space application processes query routing and result aggregation. With eBPF, we may build an in-kernel proxy that works at the XDP hook, which resides in the kernel driver layer. Compared to the user-space processing, in-kernel processing provides lower latency and high throughput by avoiding networking stack traversal overheads.

Specifically, by designing an eBPF program, we can track the status of ongoing result aggregation and route queries efficiently without traversing the host networking stack. However, several technical challenges arise. First, eBPF does not support floating-point operations, which are commonly required for similarity computations. To address this, alternative approaches such as vector quantization may be used to approximate computations using integer arithmetic. Second, eBPF enforces static memory allocation for safety reasons. Therefore, an efficient mechanism is required to manage pre-allocated memory regions for accessing vector data.

3 CONCLUSION

Vector search has become a cornerstone of AI applications, including large language models and retrieval-augmented generation (RAG), making distributed vector databases indispensable. However, existing systems suffer from broadcast and aggregation bottlenecks. We discussed a global indexing approach built on approximate nearest neighbour (ANN) techniques combined with an early termination strategy, which routes queries only to the necessary shards.

ACKNOWLEDGEMENT

This research was sponsored by the National Research Foundation of Korea (NRF) grants funded by the Ministry of Science and ICT (No. RS-2025-00522990). Gyuyeong Kim is the corresponding author.

REFERENCES

[1] 2024. eBPF. https://ebpf.io/. (2024).

[2] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei, and Charles Xie. 2021. Milvus: A Purpose-Built Vector Data Management System. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 2614–2627. https://doi.org/10.1145/3448016.3457550

[3] Zili Zhang, Chao Jin, Linpeng Tang, Xuanzhe Liu, and Xin Jin. 2023. Fast, Approximate Vector Queries on Very Large Unstructured Datasets. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 995–1011. https://www.usenix.org/conference/nsdi23/presentation/zhang-zili

[4] Zili Zhang, Fangyue Liu, Gang Huang, Xuanzhe Liu, and Xin Jin. 2024. Fast Vector Query Processing for Large Datasets Beyond GPU Memory with Reordered Pipelining. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 23–40.