

VectorDB Benchmarking

Objective

Which vector database is best for our use case? Here's a comprehensive benchmarking document.

Methodology

There are many constraints when we have to choose the right vector database and based on our requirements I have created a test environment and this environment is the same for each vector database.

- The Sample dataset has 500K embeddings of 1536 (Open AI standard) vector dimension size.
- This is a search performance test.
- The vector dbs are deployed locally as docker containers, having the following configuration. RAM: 8 GB, vCPU: 8
- The distance metric logic used is COSINE, and the vector embedding indexing logic is HNSW. Reference Document, <https://www.pinecone.io/learn/series/faiss/hnsw>
- This test has been done on same vector indexing logic, i.e. HNSW and it is not a comparison of different vector embedding logics. Reference Document <https://weaviate.io/developers/weaviate/concepts/vector-index>
- The benchmarking logic used is the same for every db. Reference Document <https://github.com/zilliztech/VectorDBBench>
- The value of k (No. of embedding retrieved) is fixed and the same for every db.
- Recall is calculated at Recall@100 (Recall@k and k is 100)
- ef parameter is variable and has been changed accordingly to increase the recall value.
- Single node of each db is deployed during load testing.

Terminology

To understand the result of the test first we have to know the basic terms used.

QPS (Query Per Second)

A vector database's capability to handle concurrent queries per second. *Higher QPS values indicate better vector database performance.*

Load Duration

This is the time taken to load the embeddings to the vector database. *Lower value of load duration indicates better vector database performance.*

Note: This is a search performance test and the load duration recorded is for a single run and may vary in different scenarios. To get a better insight of load duration a load performance test can be performed.

Memory Utilized

Memory utilization is divided in three parts, **without embeddings**: when the vector db has zero collection, **with embeddings**: when 500K dataset has been fully uploaded to the database, **Full Load(max)**: the maximum memory occupied by the db during the test. Memory is always an issue in deployment of any application. So, *Lower memory utilization is better.*

Recall Value

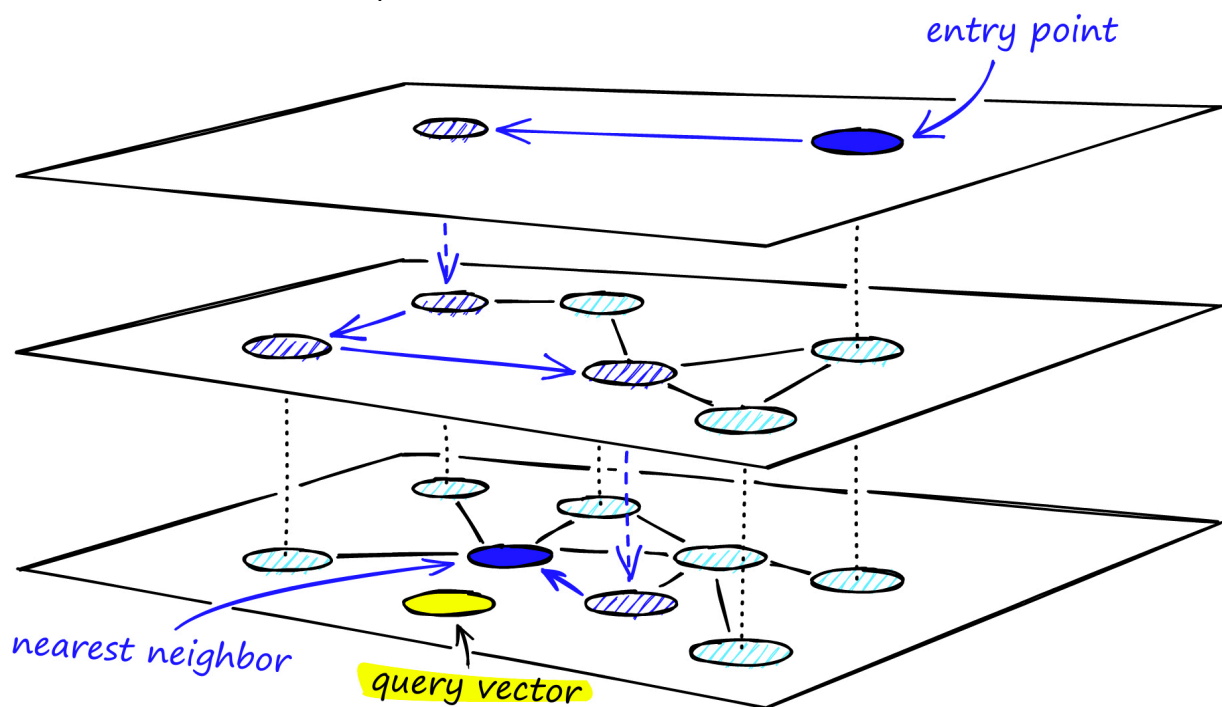
Vector search accuracy of a vector database. This is the comparison between the embeddings of exact-neighbor-search and approximate-neighbor-search on a given value of k (no. of chunks). Here it is calculated at $k = 100$. *Higher recall rates correspond to more accurate vector search results.*

efConstruction

To understand ef-construct, let me first explain how HSNW works. HNSW is a natural evolution of NSW, which borrows inspiration from hierarchical multi-layers from Pugh's probability skip list structure.

Adding hierarchy to NSW produces a graph where links are separated across different layers. At the top layer, we have the longest links, and at the bottom layer, we have the shortest.

During the search, we enter the top layer, where we find the longest links. These vertices will tend to be higher-degree vertices (with links separated across multiple layers), meaning that we, by default, start in the *zoom-in* phase described for NSW.



We traverse edges in each layer, greedily moving to the nearest vertex until we find a local minimum. At this point, we shift to the current vertex in a lower layer and begin searching again. We repeat this process until finding the local minimum of our bottom layer — *layer 0*. As you can see in the image it is pointed as the nearest neighbor.

Graph construction starts at the top layer. After entering the graph the algorithm greedily traverses across edges, finding the ef-nearest neighbors to our inserted vector q — at this point $ef = 1$.

After finding the local minimum, it moves down to the next layer (just as is done during search). This process is repeated until reaching our chosen insertion layer. Here begins phase two of construction.

*The ef value is increased to **efConstruction** (a parameter we set), meaning more nearest neighbors will be returned.*

In short, the higher the efConstruction value the denser the vector be, which will result in higher number of nearest neighbors.

Benchmark Result Analysis

The detailed result sheet can be found here [Vector DB Benchmarking](#) The important points observed from the result sheet are here,

- Qdrant DB is the best performing among all, it has the highest QPS, Highest Recall@100 (at efConstruct 256)
- Vespa and ElasticSearch db are more refined on the memory utilization side, however their latency is on the higher side.
- Load duration is the lowest in Vespa db.
- efConstruct directly impacts vector db search output like, QPS, Recall, and Load Duration.

Conclusions

Depending on the dataset size and the desired quality, some dbs may be preferable than others. Here are some of my conclusions:

- For our use case Qdrant would be the best, it has higher QPS and recall value for a given efConstruct.
- Varying nodes and ef value impact largely on vector database performance.
- When the embedding count is low (less than 100K) the recall value does not vary significantly on even lower ef values, so we can use the lower ef value which will result in higher QPS.
- Memory is an issue in Qdrant, Weviate and Chroma, since it stores the cached data, so it might need manual restarting after a time interval.