Building a Distributed Vector Database System

CSCI 598 Final Project Report

Nick Landry

Introduction

For this project, I built a working prototype vector database system with separate computation and storage nodes to support horizontal scalability. I was inspired to pursue this problem after having the brief opportunity to work with Pinecone (a vector DB) on a project for Advanced Software Engineering (Undergraduate Field Session). In this project, I had the opportunity to gather information from city websites to ultimately help an LLM understand local municipal codes. From this experience, I became interested in using a vector DB for personal tasks.

I envision utilizing this project in the future to implement retrieval augmented generation (RAG) with content I have created. This will be used to enhance the capabilities of an LLM in assisting me with studying for an exam. To accomplish this, I will connect my vector DB system into a pipeline that uses RAG to provide an LLM of my choice, such as a local LLM like DeepSeek or a cloud LLM like ChatGPT, with additional context. The vector DB system would contain relevant material I created, such as lecture notes, or homework assignments. On the surface, the vector DB system may not require a separation of computation and storage, however large amounts of training data (such as every note I have ever taken) could result in a scenario where it makes sense to rely on a separate storage node. Additionally, a remotely accessible storage node would be preferred to storing all information on limited laptop “disk” storage.

As cloud computation continues to grow in relevance, it has become necessary for database systems to separate computation and storage nodes to achieve demands caused by scale. With relational databases, I believe sharding strategies are relatively easier, given that there are clear data structures that can be placed within shards, such as putting entire tables inside single shards. However, with a vector DB, data is generally stored in an unstructured index/document store. Sharding strategies are arguably less clear, which I discovered in the process of this project.

Background and Related Work

Regarding specific resources, I found the following to be helpful in the process of researching and implementing this project:

- Facebook AI Similarity Search (Faiss), a Meta library for efficient similarity search and clustering of dense vectors [1], is designed to quickly determine nearest neighbors in high-dimension spaces. Faiss contains several similarity search methods including Euclidian distance and dot product. Faiss was relevant to the secondary version of this project, which pushed additional computation to each storage node to use traditional vector DB infrastructure.

- Discussion of Inverted File Indexes (IVF) [2] was helpful for determining a strategy for assigning vectors to storage shards. The reference page discusses how K-means clustering can be used to identify clusters in vectorized results. By identifying cluster centroids, vectors can be assigned to a shard based on distance to centroid (a vector is assigned to the shard with the centroid it is closest to). When querying for nearest neighbors, the algorithm only needs to check a small subset of shards with centroids relatively close to the query vector.
- Retrieval-Augmented Generation for Knowledge-Intensive NLP Taks [3], the foundational paper for modern RAG, discusses how LLMs often struggle with knowledge-intensive tasks. RAG combines pre-trained models with non-parametric memory to generate better results based on a larger amount of highly relevant knowledge.
- Sentence Transformers embedding models were used for generating vectors of text. Specifically, the all-mpnet-base-v2 [4] model and the all-MiniLM-L6-v2 [5] model were used to generate 786-dimensional and 384-dimensional vectors, respectively. This specific model was chosen due to its lightweight performance, since it is only designed for shorter texts (small paragraphs), making it useful for creating a working prototype. Future uses of this project will likely rely on higher dimension embedding models.

Technical Approach

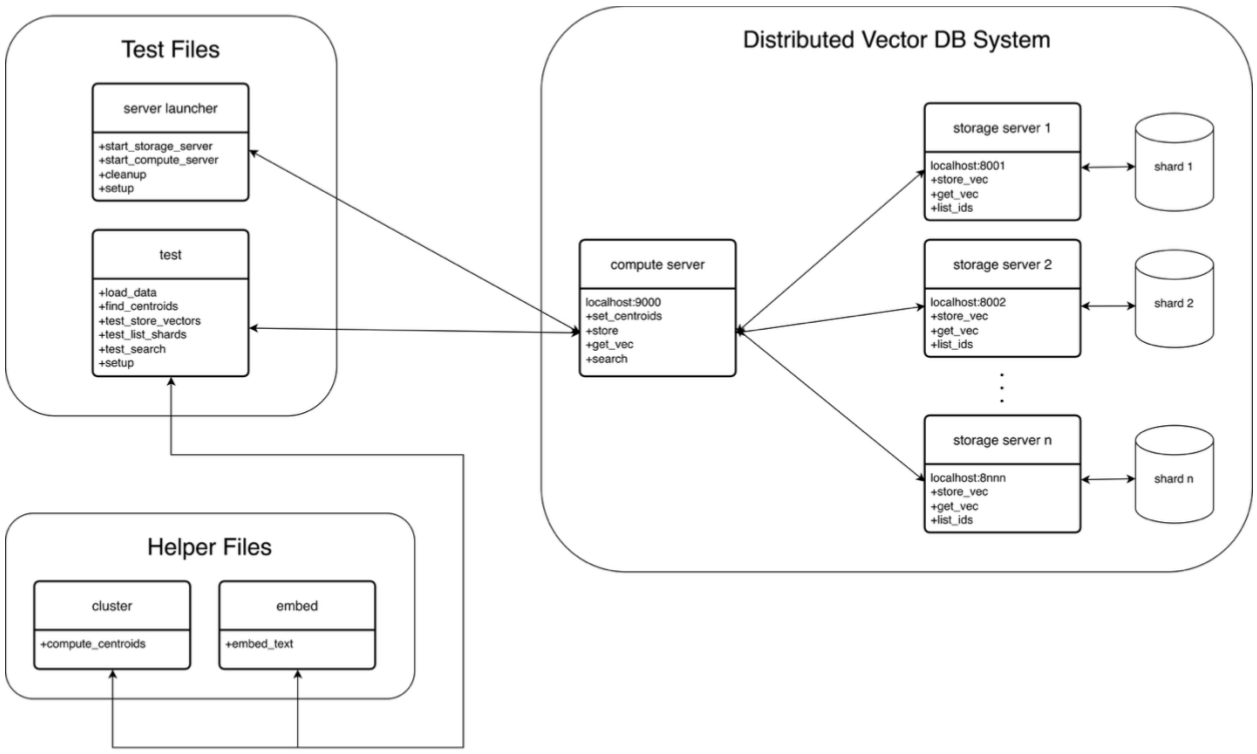

Figure 1: Project Architecture Diagram

As shown in Figure 1, the project was implemented using a single compute server and a storage server per each shard, allowing the system to scale horizontally. Within the system itself, the compute server is responsible for managing all storage servers while storage servers responsible for direct operations into each shard file. All servers can be interacted with using the corresponding HTTP endpoints for each action (ex: PUT store_vec). Server operations were implemented using Python FastAPI libraries, allowing for multiple storage servers to be created using a single python file.

In the main version of the project, data is stored using SQLite, through this could change in future iterations of the project depending on use. An additional version of the project was created using the Pymilvus library [6] to leverage quick searching using a true vector DB ANN search. In this version, each storage node is responsible for determining its K nearest neighbors, while the compute server sorts through all found nearest neighbors to determine the true K nearest neighbors. This alternate version leverages speedups from vector DB libraries at the cost of moving additional computation to the storage node. Since the primary goal of this project is to keep storage and computation as separate as possible, this version remained the alternate implementation.

A setup script was created to run setup operations – primarily server and DB file creation. This script is designed to handle all initialization and cleanup operations for compute and storage nodes. In a deployment using multiple machines, this process would need to be modified to account for setup across devices. A test script was also created to handle dataset loading and system testing. This script loads in a dataset of over one million news articles [7] for meaningful benchmarking and clustering.

Figure 1 also includes helper files, both of which perform simple operations needed for the system. The cluster file is responsible for using the K nearest neighbors clustering algorithm to identify centroids for each shard. The embed file contains code to produce vector embeddings of input text using all-mpnet-base-v2 or all-MiniLM-L6-v2.

Evaluation and Demonstration of Use

Evaluation

As mentioned above, a dataset of over one million news articles [7] was used for benchmark testing. The benchmarking code loads in the first 10,000 articles and produces KNNs (where k is the number of shards), thus requiring KNN to be performed for each unique embedding and shard combination. Despite the limited dataset of 10,000 articles, which may seem small, the results were still meaningful, as discussed in further detail below.

A single query "Historical wars in Europe." was used for all testing given that it returned relevant results in initial testing. All test queries were prompted to return the top 5 nearest neighbors. Two embedding algorithms, all-mpnet-base-vs2 with 768-dimensional vectors and all-MiniLm-L6-v2) with 384-dimensional vectors, were used to produce a simple explanation of the effect of dimension size on runtime. This project was not intended to be a benchmarking project, so further query or embedding tests were deemed to be beyond the scope and limited time frame.

Given the purpose of the project is to integrate the vector DB system with a RAG data pipeline to provide an LLM with additional material, runtime was deemed important. As an end user, I would not want to use the system if I found its search execution times to be too slow. Thus, query runtime performance was the primary analysis metric used below.

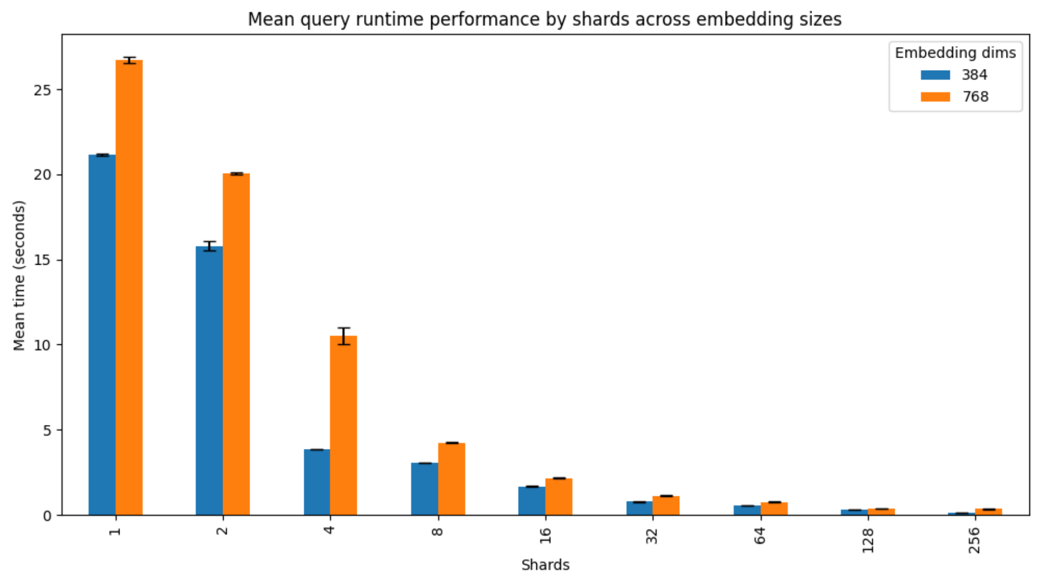

Figure 2 (Above): A graph (standard y-axis) showing query runtimes across different sharding sizes for two embedding sizes.

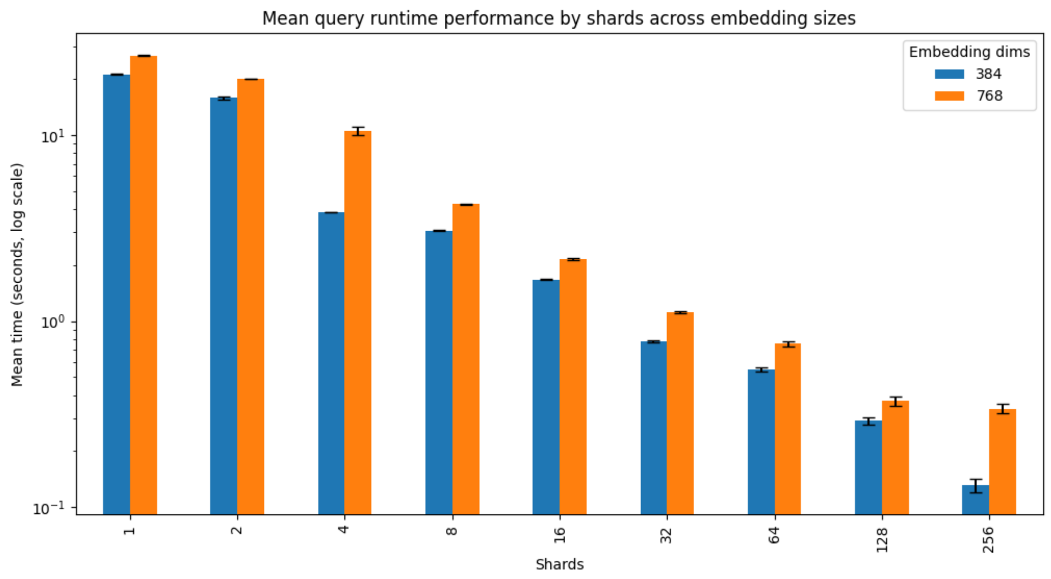

Figure 3 (Above): A graph (log y-axis) showing query runtimes across different sharding sizes for two embedding sizes.

A quantitative evaluation was generated by measuring the average wall clock time for a common query at different sharding levels for two different embedding dimensions. As show in the graphs (Figure 2 and Figure 3) above, query performance improved for both embedding dimensions as the number of shards increased. This is not surprising, given the average size of each shard should decrease as the number of shards increases. Since only a few shards are considered on each query, decreasing the shard size should reduce the number of total nearest neighbor computations. Increasing the shard size beyond 256 resulted in memory errors, thus 256 was the highest chosen metric. It is expected that continuous shard count increases would eventually result in worse performance as the overhead of handling each shard becomes more costly than the very small remaining gains from having a few (or possibly 0-1) entries in each shard.

Demonstration of Use

The following walkthrough demonstrates the effectiveness of the project.

First, the user should specify the number of shards and embedding dimension to use in a .env file (the dimension should correspond to the chosen embedding algorithm in the embed helper file). The following variables were used for this example:

```
NUM_SHARDS = 64
EMBED_DIM = 786
```

Once all variables are specified, the user should run the helper script to start all compute and storage server nodes. The script will create a file for on-disk data storage (in the /data folder) and a HTTP server for each storage node, and an HTTP server for the compute node. Once all servers are running, the user is given a helpful confirmation message (with a clearly AI-generated emoji):

```
 ==============================================
🚀 Servers are running!
Press CTRL+C to stop everything.
==============================================
```

At this point, the user should specify remaining parameters in the test file, of which 'top_k' (how many results should be returned) and 'query' should be specified. Below are the example variables used:

```
top_k = 5
query_str = "Historical wars in Europe."
```

Once the code is run, the user will be provided a series of messages indicating the stage of the process. A large portion of the process is the initial dataset loading/vectorization and the insertion of all inputs to their respective shards using POST requests. After the search process completes, the user is provided with results. Below are the results from this sample test:

```
Search results:
{
  "results": [
    {
      "id": "Darfur #39;s War of Definitions Finally, the conflict in Darfur in western Sudan is a focal point in international diplomacy and media attention. This is the least to expect after months of bloody campaigns of murder, rape and dare I say, ethnic cleansing, starting as ...",
      "score": 0.41488136776344653,
      "shard": 43
    },
    {
        "id": "EU nations slam Tiger rebels as fresh killings dim Sri Lanka peace hopes (AFP) AFP - European Union nations criticised Sri Lanka's Tamil Tiger rebels for killing rivals and recruiting child soldiers, appealing to the guerrillas not to undermine the Island's Norwegian-led peace bid.",
        "score": 0.40409126592148625,
        "shard": 43
    }
```

(results are shortened for space, the full result is available in the appendix)

The results above show that the system can successfully produce news article texts using nearest neighbor search, thus successfully achieving the overall goal of the project.

Generative AI Disclosure

Generative AI was used rather extensively throughout this process. From the start, I use ChatGPT to suggest a plan for this project, starting from a single compute and storage node. Additionally, I used ChatGPT to generate starter code for many of the files I relied on, and to write tests to ensure the system functioned as expected. I also used Codex (in VSCode) to debug error messages I received, since I found it to be more effective given that it has all the necessary context (current versions of every file). Codex performed reasonably well at identifying the source of errors, often making the correct suggestion the first or second time.

However, I made the initial mistake of attempting to “vibe code” the entire project, which I quickly discovered was not feasible. ChatGPT generated overly complex code designed more for a production-ready environment rather than a class project. I struggled to understand the code, ultimately making the decision to start from scratch by creating a plan (with ChatGPT) before writing any code. Additionally, I found Codex to quickly attempt to solve errors with lackluster explanations and sometimes incorrect fixes. I decided to fix bugs myself but found Codex to be helpful in identifying error sources.

Future Work and Conclusion

The most pressing future work involves integrating this project with a RAG data pipeline to feed information to an LLM. As discussed in the introduction section, use cases include improved study help, where an LLM can get information from specific problems that a query encompasses using RAG. Additional future work could involve improvements to the embedding algorithms to more effectively encompass larger samples of text. This could also include embedding non-text data, such as images. Furthermore, the project could add an optimization feature to perform calculations that automatically determine the number of shards to use based on system information and clustering results.

Additional evaluations should attempt to measure the effectiveness of the system's vector search to return relevant results for the LLM. Once integrated with an LLM, testing could assess the helpfulness of RAG to improving LLM results. Both sets of tests would be relatively subjective given how the user defines results to be "relevant" or an LLM to be "helpful".

My favorite part of the project was the planning of interactions and responsibilities of each node. I found it challenging to balance the specific responsibilities of each node with overall performance, since moving certain computations (primarily ANN) to storage nodes would improve the performance of the project at the cost of scalability. For these reasons, I created two versions of the project and chose to discuss the slower, more scalable version (since scalability was my overall goal).

Over the course of this project, I gained additional experience using Vector DB systems, HTTP endpoints/servers, and architecture planning. Beyond technical experience, I also learned how tradeoffs can directly affect DB performance. Since this project was open-ended, I found myself spending more time than anticipated on planning and coding implementation. This was because I was frequently coming up with ways to enhance the project.

Citations

[1] Facebook Research, “Faiss,” GitHub, Jan. 12, 2023. https://github.com/facebookresearch/faiss

[2] “How do inverted file (IVF) indexes work in vector databases, and what role do clustering centroids play in the search process?,” Milvus.io, 2024. https://milvus.io/ai-quick-reference/how-do-inverted-file-ivf-indexes-work-in-vector-databases-and-what-role-do-clustering-centroids-play-in-the-search-process

[3] P. Lewis et al., “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” Apr. 2021. Available: https://arxiv.org/pdf/2005.11401

[4] “sentence-transformers/all-mpnet-base-v2 · Hugging Face,” huggingface.co. https://huggingface.co/sentence-transformers/all-mpnet-base-v2

[5] “sentence-transformers/all-MiniLM-L6-v2 · Hugging Face,” huggingface.co. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

[6] “Run Milvus Lite Locally | Milvus Documentation,” Milvus.io, 2025. https://milvus.io/docs/milvus_lite.md

[7] S. Lee, “ag_news,” Huggingface.co, 2015. https://huggingface.co/datasets/sh0416/ag_news

Appendix

Code

Although not required for this project, all code is available in a public GitHub repository.

Full Query Result

Search results:

```
{
  "results": [
    {
      "id": "Darfur #39;s War of Definitions Finally, the conflict in Darfur in western Sudan is a focal point in international diplomacy and media attention. This is the least to expect after months of bloody campaigns of murder, rape and dare I say, ethnic cleansing, starting as ...",
      "score": 0.41488136776344653,
      "shard": 43
    },
    {
      "id": "EU nations slam Tiger rebels as fresh killings dim Sri Lanka peace hopes (AFP) AFP - European Union nations criticised Sri Lanka's Tamil Tiger rebels for killing rivals and recruiting child soldiers, appealing to the guerrillas not to undermine the Island's Norwegian-led peace bid.",
      "score": 0.40409126592148625,
      "shard": 43
    },
    {
      "id": "US troop shift: A tale of two cities Bush's plan this week to restructure US forces in Europe is stirring up a mix of worry, nostalgia, and hope.",
      "score": 0.3975063509739522,
      "shard": 43
    },
    {
      "id": "THE UNITED NATIONS The Campaign against Kofi The secretary general of the United Nations fights a war on many fronts in his crusade to bring human rights and peace to the world.",
      "score": 0.3804943480130972,
      "shard": 43
    },
    {
      "id": "Western leaders waiting for African solutions to African wars JOHANNESBURG, South Africa, Aug 19, 2004 (AP) -- As the month-end deadline nears for Sudan to disarm the Janjaweed militias in Darfur, the United Nations and Western powers debate how far to go to stop the killing ...",
      "score": 0.36443419414908274,
      "shard": 43
    }
  ]
}
```