Homework_4
Team_11
Q2 Distributed k-NN System using gRPC
Detailed Report

Vilal Ali, 2024701027, (Questions 2 and 3)

Kapil Rajesh Kavitha - 2021101028 (Questions 1 and 4)

1. Overview

The goal of this project is to implement a distributed k-Nearest Neighbors (k-NN) system using gRPC for communication between multiple servers. Each server holds a subset of a dataset and, when queried, calculates the k-nearest neighbors for a given data point. The dataset is partitioned across these servers, which act as distributed workers. The servers communicate with the client via gRPC, facilitating efficient, scalable, and distributed computing.

The system supports multiple gRPC servers that work in parallel, each responsible for a subset of the dataset. The client communicates with an aggregator, which in turn queries the individual dataset servers, aggregates their results, and returns the final k-nearest neighbors to the client. The core of the system uses Euclidean distance for k-NN calculation.

2. Key Features

- Distributed Data Handling: Each server holds a portion of the dataset, reducing the load on a single machine and enabling parallel computation.
- gRPC Communication: gRPC is used to facilitate efficient, bidirectional streaming communication between the client, aggregator, and dataset servers.
- Fault Tolerance: The system is designed to handle failures by ensuring that each server operates independently on its subset.
- Scalability: The system can scale by adding more dataset servers to handle larger datasets or increasing the number of servers for better load distribution.

3. Architecture and Components

1. **Client:**
   - Sends queries to the aggregator, specifying a data_point and k value.

- Receives the final result of the k-nearest neighbors from the aggregator.

2. **Aggregator:**
   - Receives the client's query and forwards it to all dataset servers.
   - Gathers responses from the dataset servers, merges the results, and returns the final set of nearest neighbors to the client.

3. **Dataset Servers:**
   - Each server holds a portion of the dataset, computes the k-nearest neighbors from its subset, and returns the result to the aggregator.
4. **gRPC Protocol:**
   - gRPC is used for communication between the client, aggregator, and servers. It allows efficient data transmission using Protocol Buffers (protobuf) as the serialization format.

4. gRPC in Distributed Computing

gRPC facilitates distributed computing by providing the following advantages:

- **Interoperability:** gRPC allows communication between different services across various languages and platforms. This is crucial for distributed systems where components might be implemented in different programming languages or run on different architectures.

- **Efficient Streaming:** gRPC supports both unary (request-response) and streaming (bidirectional) communication. In our system, gRPC allows the aggregator to concurrently communicate with multiple dataset servers, ensuring that responses are gathered in parallel for faster aggregation.

- **Low Latency and Scalability:** gRPC uses HTTP/2, which enables multiplexing of streams, header compression, and other optimizations, resulting in lower latency and better scalability compared to traditional HTTP/1.x communication.

- **Error Handling and Status Codes:** gRPC provides built-in mechanisms for handling errors and communicating them back to the client using standardized status codes. This helps in building robust distributed systems with clear error reporting.

5. Comparison between gRPC and MPI

Aspect	**gRPC**	**MPI (Message Passing Interface)**
Communication Model	Client-server model, where clients make requests to servers (RPC-based).	Peer-to-peer communication model, where processes explicitly send and receive messages (message-passing).
Usability	Easier to use for building distributed systems, particularly in cloud and microservices.	Requires deeper understanding of parallel computing concepts, generally used in HPC environments.
Data Serialization	Uses Protocol Buffers (protobuf), which is efficient and language-agnostic.	Does not have built-in serialization; developers need to manage data marshalling/unmarshalling.
Scalability	Highly scalable due to its support for HTTP/2, multiplexing, and load balancing.	Scalable for tightly coupled parallel applications but requires significant effort for dynamic scaling.
Error Handling	Built-in status codes and error handling, simplifying development.	Requires manual error handling, often leading to more complex code in large-scale systems.
Environment	Best suited for microservices, cloud-native architectures, and distributed web applications.	Primarily used in high-performance computing (HPC) for scientific simulations and computations.

Conclusion:

- gRPC is easier to use and better suited for distributed systems that involve microservices, cloud, or hybrid architectures.
- MPI, while powerful, is more complex to implement and is best for tightly coupled, highly parallel computations typically found in HPC environments.

6. Dataset Partitioning

The dataset is split into parts and distributed across multiple gRPC servers. Each server operates on its own subset of data and computes the k-nearest neighbors locally. The aggregator consolidates the results from all servers, ensuring that the final k-nearest neighbors are computed across the entire dataset.

Example Partitioning Logic:

- Given a dataset D of size n, and m servers, each server receives approximately n/m elements of the dataset.
- Each server operates on its assigned partition and, when queried, only searches within this subset.

7. Distance Calculation: Euclidean Metric

The system uses **Euclidean distance** as the metric for calculating the nearest neighbors. The formula for Euclidean distance between two points p and q in a d-dimensional space is:

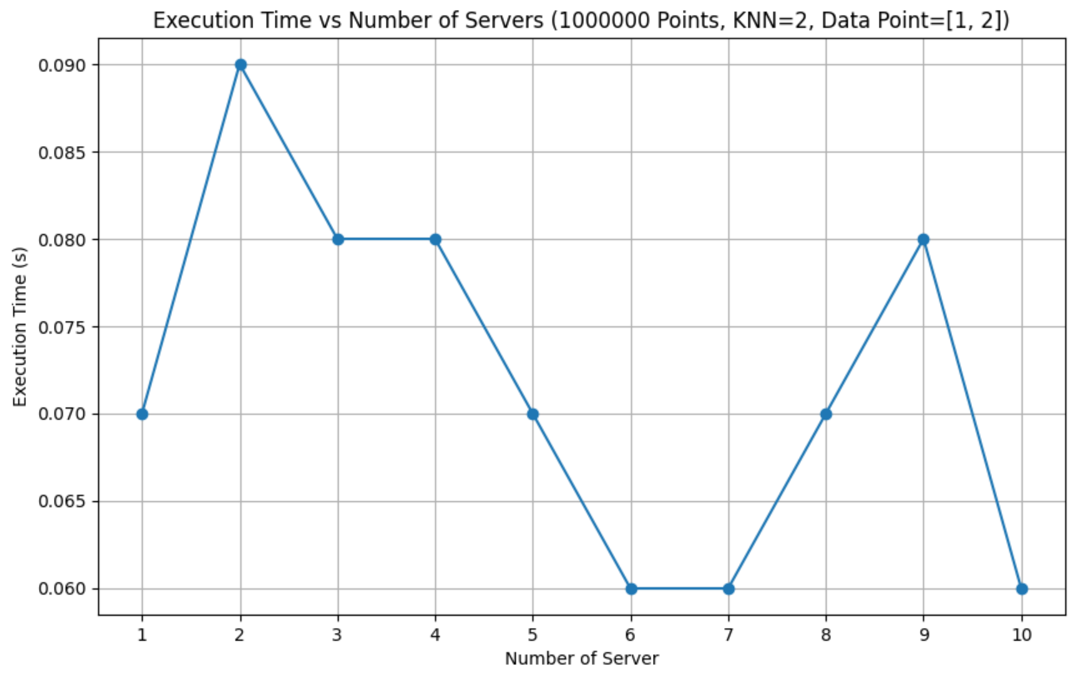
$$d(p,q) = \sqrt{\sum_{i=1}^{d}(p_i - q_i)^2}$$

This metric is simple and computationally efficient, making it well-suited for this system, where multiple servers compute distances concurrently.

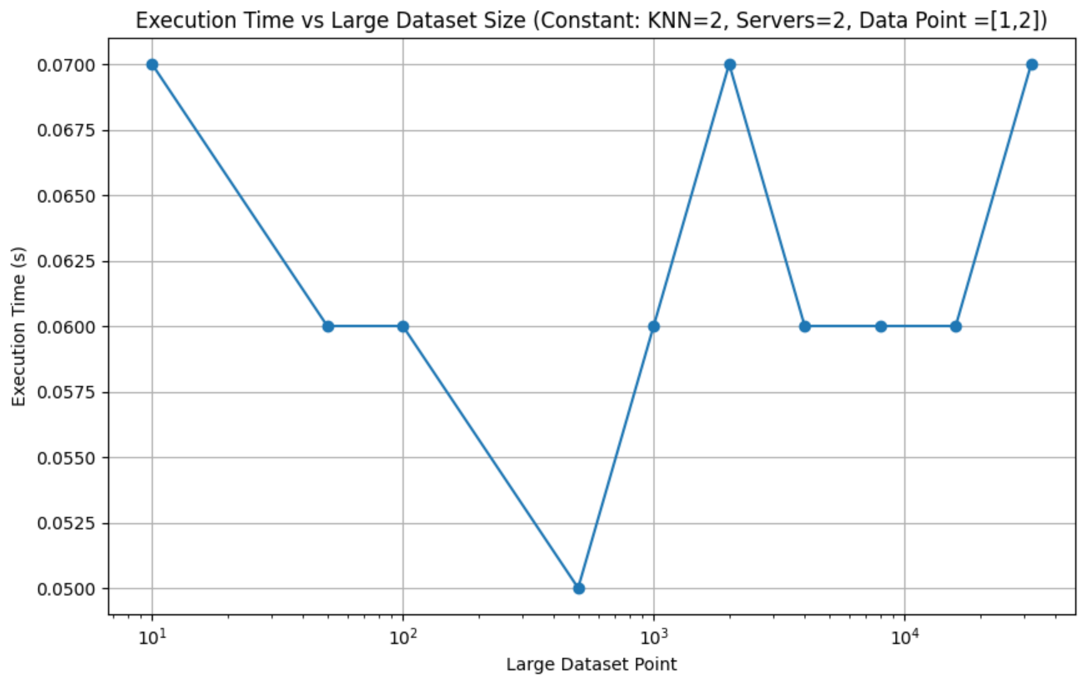
8. Performance Analysis

The performance of the distributed k-NN system was analyzed using different dataset sizes and varying values of k. The following factors were considered:

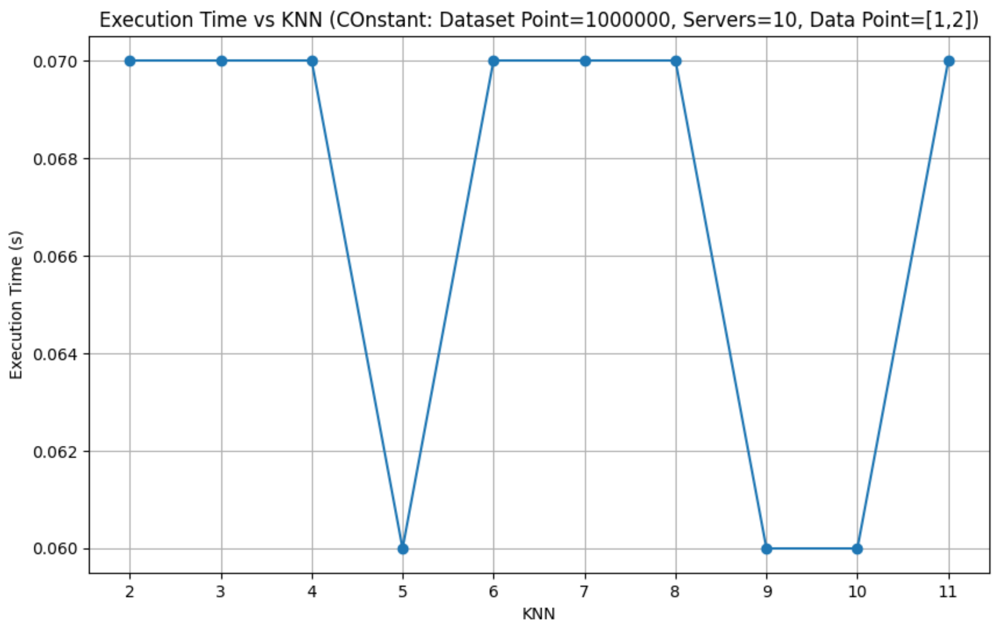
Performance Evaluation: Impact of Increasing the Number of Servers on KNN=2 with 1,000,000 Data Points for Data Point [1, 2]



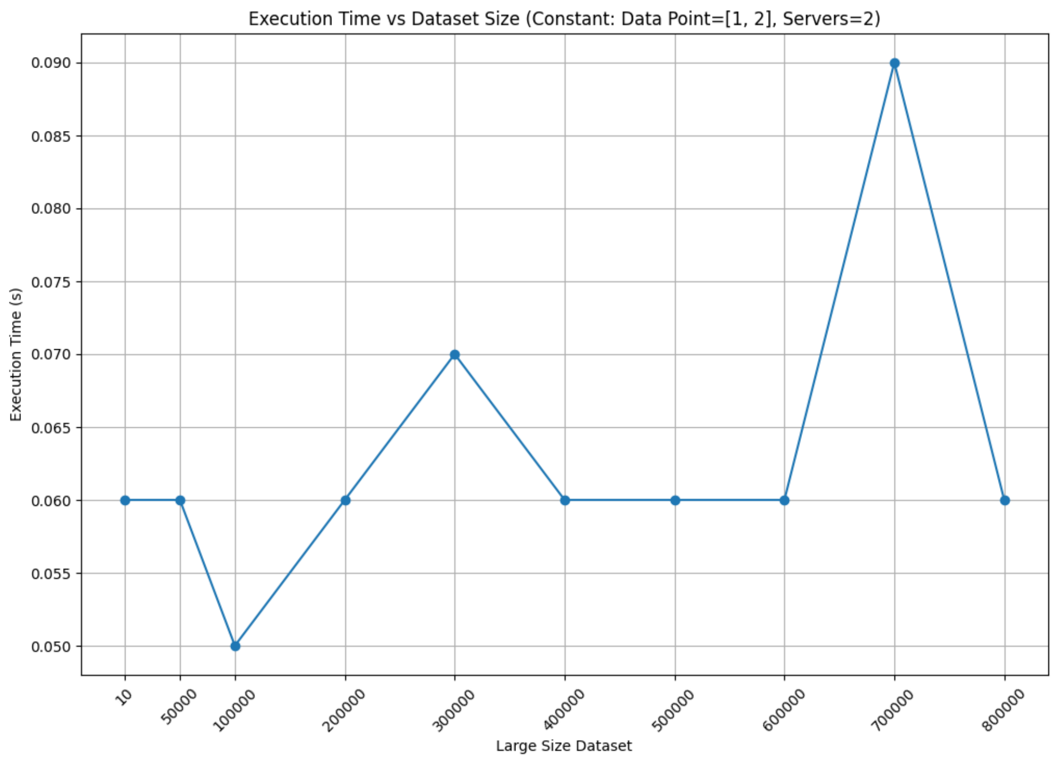
Performance Evaluation: Effect of Changing Dataset Size on KNN=2 with Data Point [1, 2] Using 2 Servers



Performance Evaluation: Influence of Increasing KNN on Performance with 1,000,000 Data Points and Data Point [1, 2] Using 10 Servers



Performance Evaluation: Variation of KNN and Dataset Size with Data Point [1, 2] Using 2 Servers



9. Conclusion

This project successfully demonstrates the application of gRPC for building a distributed k-NN system. By leveraging gRPC's low-latency communication and efficient data serialization, the system achieves scalability and high performance even with larger datasets and higher values of k. Additionally, the comparison between gRPC and MPI shows that gRPC is better suited for this type of distributed system due to its ease of use, scalability, and support for microservices-like architectures.

Note:

Project Setup and Execution: Instructions for running the project and configuring the environment can be found at **Team_11/Q2/README.md**.

Use Case and Demo Video: The demo video showcasing the use case can be found at **Team_11/Q2/Q2_Demo.mp4**.