Q-2 (HW-4)

K-Nearest Neighbors

Report

Akshit Sharma
2021101029

Efficiency Considerations

Instead of querying each server sequentially, they have been queried in parallel (by using a thread to query a server, while other threads query the other servers, to save time). Also, a single RPC is used to send all the local kNN from a server to the client, instead of each point (and its distance from query point) individually.

gRPC facilitates distributed computing

gRPC facilitates distributed computing in this k-NN implementation by providing a communication framework that allows the client and servers to interact efficiently across distributed systems. This can be explained as follows:

Client-Server Interaction

- Client Request: The client initiates the request by sending a query point and the value of **k** to each server. This is done using gRPC's **remote procedure calls (RPCs)**, which allow the client to communicate with distributed servers over the network as if they were local function calls.
- Server Response: Each server computes the local k-nearest neighbors from its subset of the dataset and sends the results back to the client using gRPC's response mechanism.

Parallel Processing

- Since each server holds a portion of the dataset, the computation of k-nearest neighbors happens in parallel across multiple servers. gRPC enables these servers to independently handle the client's request in parallel without waiting for each other.
- The gRPC framework handles **asynchronous communication**, allowing the client to send requests to multiple servers simultaneously and receive results as they are computed. This reduces the overall computation time.

Client-side Aggregation

- Once the client receives the local k-nearest neighbors from each server, it pools these local results together. Using a **heap** at the client, the results are merged efficiently to compute the global k-nearest neighbors.
- gRPC's ability to handle large volumes of communication efficiently ensures that this aggregation can happen without significant delays or network overhead.

Load Balancing and Scalability

- gRPC allows multiple servers to handle portions of the dataset in a scalable fashion. This helps in distributing the computational load and achieving faster responses.
- The distributed nature of gRPC makes it easy to add more servers or scale up the system by adding more resources to handle larger datasets or more queries.

Comparison between gRPC and MPI

Communication Model

gRPC (Client-Server Model): Follows a loosely coupled client-server architecture where the client sends requests to independent servers, and servers respond asynchronously. Communication happens over HTTP/2, enabling bidirectional streaming.

MPI (Tightly-Coupled Message Passing): Uses a tightly-coupled message-passing model, where processes communicate directly in a peer-to-peer or master-slave structure. It's designed for high-performance computing (HPC), with

optimized communication in tightly connected clusters, typically using shared memory or direct sockets.

Distributed vs. Parallel Processing

gRPC (Distributed System): gRPC is optimized for distributed computing across large networks (e.g., the Internet or cloud infrastructures). Servers can be located in different places, and the client collects results asynchronously.

MPI (Parallel System): MPI focuses on **parallel processing** within tightly-coupled systems, typically using a cluster of machines with low-latency connections. Each MPI process runs on a separate node or core, sharing memory or using fast interconnects.

Scalability and Flexibility

gRPC: Highly **scalable** due to its loosely-coupled architecture, allowing for easy addition or removal of servers and deployment across different geographic regions without significant architectural changes. It includes built-in support for **load balancing**, retries, and fault tolerance, making it ideal for **cloud-based and dynamic environments**. Additionally, gRPC offers **flexibility** by supporting various platforms and languages through Protocol Buffers.

MPI: Optimized for **high-performance scalability** in HPC environments, capable of scaling to thousands of processors. However, it requires careful resource management, as adding processes or nodes often necessitates system reconfiguration. MPI is less flexible than gRPC, as it doesn't support heterogeneous systems well and is best suited for **dedicated clusters** with tightly controlled environments.

Fault Tolerance and Error Handling

gRPC: It includes **built-in mechanisms for fault tolerance**, allowing clients to retry requests or redirect them to another server if one goes down. This is essential for distributed systems prone to node failures or unreliable networks. It leverages HTTP/2 features like flow control and multiplexing, along with built-in error handling at the transport level.

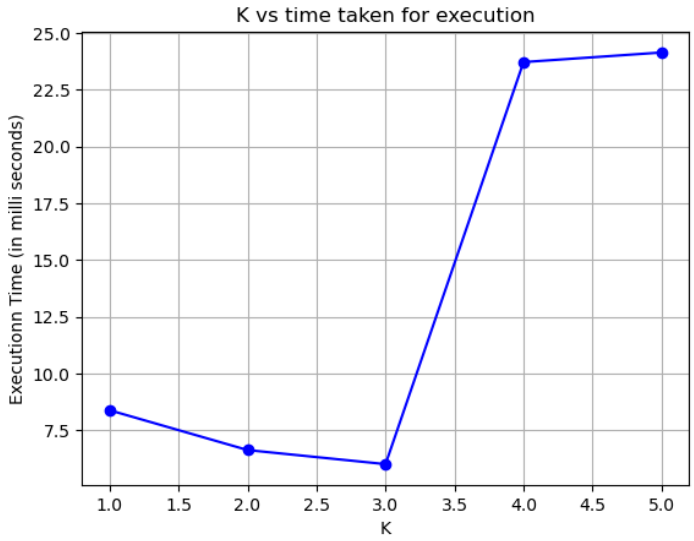
MPI: It lacks inherent fault tolerance. if an MPI process fails, it often results in the failure of the entire application. It is designed for environments where reliability is assumed (e.g., supercomputing clusters), requiring developers to implement their own error handling and recovery mechanisms.

Performance Comparison

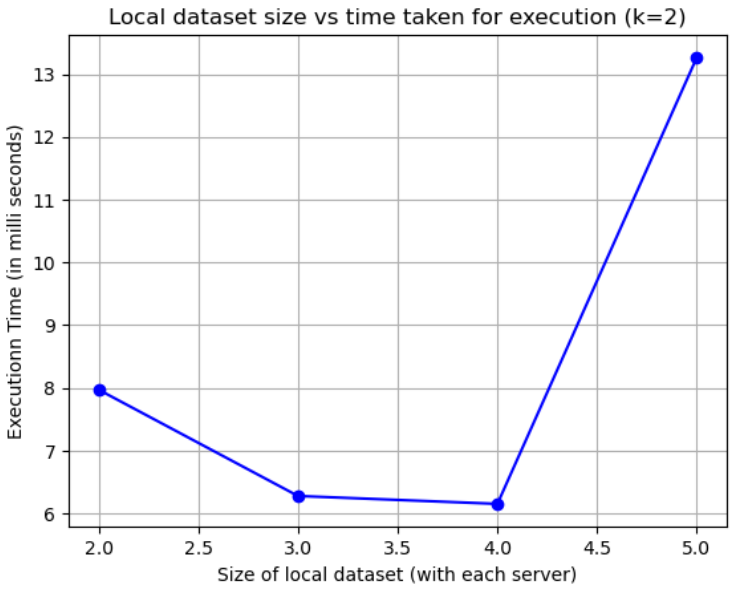
- For both increasing k and larger datasets, MPI theoretically offers better performance due to its tight coupling, lower communication latency, and efficient result aggregation. gRPC's performance will degrade faster with large k values and larger datasets due to network latency and the complexity of managing result aggregation.
- gRPC's distributed nature may lead to slower aggregation due to the geographic dispersion of servers and higher latency in communication. MPI, with its tightly-coupled processes and shared memory, provides faster local computation and more efficient result aggregation.
- gRPC offers better flexibility in terms of geographic and platform scalability (cloud, heterogeneous servers), but with increased communication overhead. MPI is designed for scaling within high-performance clusters, offering superior performance scaling in tightly-coupled systems but not as well suited for geographically distributed setups.

- For long-running computations or in unreliable environments, gRPC's fault tolerance makes it more robust. MPI, while faster and more efficient, is less resilient to failures.

Graph for value of k vs execution time



Graph for local dataset size vs execution time



Note: the program was executed on personal device. So, the time for execution may depend on the current state of hardware and other processes running on the system.