Distributed Systems

Homework-4 Question-2 Report

Kritin Maddireddy (2022101071)

December 31, 2024

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

K Nearest Neighbours

How does gRPC facilitate distributed computing in this example?

gRPC, with the help of Protobuf, lets us define all the requests, responses and procedure calls that we will be using in this program. It then produces the necessary primitives that we can use as a common message passing system for both the client and the server.

Firstly, both the client and the server are asynchronous, as they would need to handle multiple servers/clients (respectively) simultaneously, without blocking for any one of them.

Now, gRPC provides a CompletionQueue that we can use to keep checking the clients/servers to see if they're done processing, and if yes, get the response.

Suppose, there are *n data points*, *p servers* and *k nearest neighbours* are required for a query point. Suppose, each point is of dimension *l*, and thus, assume that Euclidean Distance calculation is in *O(l)* time.

The client first loads the dataset from a given file, and partitions it into different chunks to send them off to different servers. Each chunk is nearly equal in size, and there is at most 1 record that's different between each chunk. Each of the servers perform K Nearest Neighbours computation on their part of the data, and send ONLY THE TOP K neighbour points back to the client (partial sorting used here). The client keeps checking to see if a server is done processing, and if yes, it aggregates the results.

Once the client aggregates all the results, it partial sorts these *p*k* results and prints the top *k* nearest neighbours.

The sending of each request (with the query point, number of nearest neighbours to find and the partition of dataset) to a server, and the retrieval of the k nearest neighbours (along with their respective distances) on the chunk that the server has, back to the client is done via remote procedure calls, via gRPC.

Comparison between gRPC and MPI in terms of communication models, usability, and scalability

gRPC

1. **Communication Model:** *Loosely-coupled* client-server model, where communication is via HTTP and the requests and responses are defined via Protobuf. Designed for distributed systems over networks (say, for web services) and supports both synchronous and asynchronous communication.

2. **Usability:** Easy to use, considering the fact that all you need to do is invoke a procedure call that's essentially like invoking an API via traditional RESTful APIs. Protobuf generates the API calling code on whichever language you prefer (currently, it supports a lot of commonly used languages) and thus you can just pretend that the function you're invoking on the client-side is a black box, and implement it on the server-side later.

3. **Scalability:** Scales well in distributed and cloud environments, capable of handling millions of remote procedure calls over the internet. Speed/Latency depends on your internet connection.

MPI

1. **Communication Model:** *Tightly-coupled* message passing in distributed memory systems and requires direct communication between processes (NOT over the network). So, it is useful when high-performance parallel computing is required, and low-latency is of the utmost priority (so, it is useful on compute clusters). It supports both synchronous and asynchronous communication.

2. **Usability:** Harder to use, considering that you would need to explicitly mention all the send and receive commands. Debugging is difficult as well, since multiple clients/servers may respond at the same time for collective operations like broadcasting/scattering. However, it abstracts out the broadcast/scatter protocols which makes such cases easier to implement, as compared to gRPC where you would have to implement these yourself.

3. **Scalability:** Highly scalable in tightly-coupled environments (e.g., HPC clusters, like ADA), optimized for low-latency communication.

Performance analysis for different sizes of datasets and k

Note: *Running this on 5 servers. Not counting the time taken for the outputs to be printed; only the time taken to send dataset to servers (not including partitioning) and to get an output back, and sort the obtained output to get the top K nearest neighbours is being counted.*

For dataset of size 1M (1000000) data points

K values	Time Elapsed
10	0.669532s
100	0.717261s
1000	0.735258s
10000	0.897892s
100000	1.89544s
1000000	5.01548s

For K of size 100

Dataset size	Time Elapsed
100	0.003404s
1000	0.00661s
10000	0.015641s
100000	0.080322s
1000000	0.717261s