HPC project description

Parallel implementation of KNN clustering using MPI and parallel implementation of KNN clustering with hierarchical federated learning

The K-Nearest Neighbors (KNN) algorithm is a machine learning algorithm used for classification and regression. Its basic idea is very simple: given a point to classify or predict, KNN finds the K points closest to it in the training dataset and decides the class label based on which class is most common among the K closest points .

Specifically, the KNN algorithm follows the following phases:

1. **Training phase:** KNN stores the training dataset, which consists of examples with their class labels. Each example is represented by an attribute value.

2. **Testing phase:** given a new point to classify, KNN calculates the distance between this point and all points in the training dataset, often using Euclidean distance or other distance metrics.
3. **Selection of K neighbors:** K represents the number of points closest to consider. A common value for K is 3 or 5, but this can vary depending on the problem. Larger values of K make the algorithm more robust to noise, but can lead to confusion between similar classes.
4. **Majority Voting:** To classify a new point, KNN counts how many instances of each class are located between the K nearest points. The most common class label among K neighbors is assigned to the new point.

It is important to note that the effectiveness of KNN can be influenced by the choice of distance metric, the value of K and the size of the dataset.

However, the K-Nearest Neighbors machine learning algorithm is inherently sequential because its approach to calculating distances and selecting the closest points for classification or regression requires examining each point in the training dataset one at a time, and also , after calculating the

distances, it is necessary to select the K points closest to the query point, which implies a total scan of all calculated distances.

Our intention is to attempt a partial parallelization of this algorithm, specifically, to attempt to parallelize the calculation of distances and the ordering of such distances

- **Parallelization of distance calculation:** one of the most important parts computationally intensive of KNN is the calculation of distances between the query point and all points in the training dataset. This operation can be parallelized by dividing the dataset into parts managed by separate processes. Each process calculates the distances between the query point and a portion of the points.
- **Parallelization of distance ordering:** after calculating the distances, you need to sort all the distances to understand which ones are smaller than the query point. This operation can be parallelized using a parallel mergeSort between all processes in the cluster.

In order to implement parallelization, we opted to use the MPI (message passing interface) standard which offers a series of primitive functions aimed at exchanging messages between processes. Specifically, our solution will involve the division of the dataset into smaller chunks which will be divided between the various processes thanks to the MPI standard.

Subsequently calculate in parallel for each process (therefore for each chunk of the dataset) the distances between all the points of the dataset and the query point and sort these distances through a parallel implementation of the MergeSort sorting algorithm. The time it takes to get the rankings will be calculated and displayed.

It is possible to carry out a parallel implementation of the KNN algorithm using a federated context (federated learning). Federated learning is an approach to training machine learning models that allows you to train models on data distributed across devices or servers without having to centralize the data in a single location. Instead of collecting all the data in a single repository or server, federated learning allows you to train a model directly on individual user devices or on local servers, and only the training results (such as model weights) are sent to a central server to be aggregated and update the global model.

The KNN algorithm does not train a global model like many other algorithms. Instead, it classifies the data points by calculating the distance between the existing training points and the new data points to be classified. In a federated context, k-NN can be applied in a distributed manner, where each local device or server maintains its own training dataset and computes predictions based on that data.

An advanced federated learning approach is exploited which is Hierarchical Federated Learning which involves a hierarchical participation structure in model training; participants are therefore organized into a hierarchy of levels with different roles and responsibilities. In particular, the structure designed to implement the KNN algorithm is made up of three different hierarchical levels:

1. The central server is at the highest level and plays a role of coordination, communication management and aggregation of results.
2. Intermediate servers can be used to perform intermediate aggregation and coordinate local devices.
3. Local devices maintain training data and participate in processing.

This hierarchical multi-level structure requires particular attention in data distribution, since devices belonging to the same level have access to the same type of data

A possible implementation of the KNN algorithm in a hierarchical federated context could follow these steps:

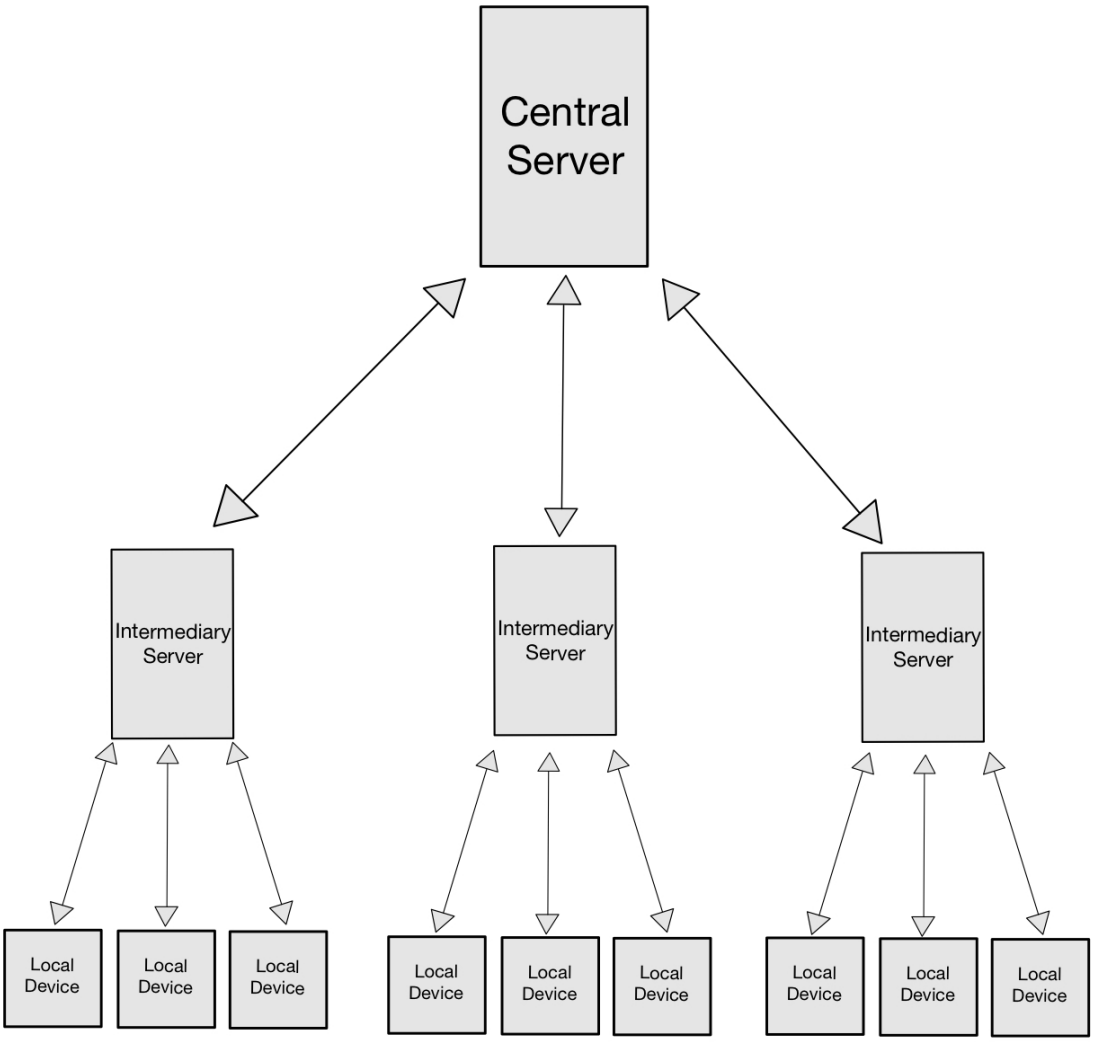
- **Data Distribution:** Each device maintains its own data set training, which is similar for each level.
- **Calculation of local KNN distances:** When a request arrives classification for a new data point, each device (independently) runs the k-NN using its own training dataset to calculate distances and thus make KNN predictions.

- **Local aggregation of results:** Subsequently each local device it aggregates the results based on its local data, simply calculating the most common class among the local predictions.
- **Communication between hierarchical levels:** At this point a communication between hierarchical levels, in particular the aggregated results from local devices in the same level are sent to the corresponding intermediate servers. Subsequently these results can be

further aggregated by intermediate servers and then be sent to the central server, or be sent directly without carrying out the aggregation.

- **Global aggregation:** The central server receives the results from the various levels hierarchical. It aggregates the results globally, for example by counting the most common classifications across all aggregated data.
- **Final classification:** The aggregate result represents the final classification of the data point.

A figure follows to make the hierarchical structure clearer



For this type of implementation it was decided to use the Python language since there is a library, TensorFlow Federated (TFF), which is suitable for federated learning.