CS633 (Parallel Computing) Assignment 3 Report

Author

Mayank Sharma (160392)

Description

In this assignment, I have implemented K-Means clustering algorithm using MPI to parallelize the problem of clustering. A brief description of the algorithm is as follows:

1. The root node is responsible for reading the data file firstly. It then scatters if over to all the other nodes. The reason I have chosen to go with this method is that the algorithm that I've used relies on the root node having a copy of full data set later to label which cluster each points belongs to.
2. The value of K will be supplied to the program externally via a command line argument. I've limited the number of iterations performed to 200 (which is normally sufficient for K-means to converge). See this link
3. During the pre-processing step, I'm firstly diving the number of points equally to each process.
4. The initial centroids have been randomly picked and are BCast from the root node to all the processes.
5. Each node uses its set of points and the broadcasted `local_centroids` to assign each point some cluster. Then, I have used a custom `MPI_Datatype` for encapsulating my own struct and then performed `MPI_Gather` to gather these `local_centroids` on the root node.
6. The root node then re-computes the new centroids and in the next iteration, broadcasts these newly computed centroids over to all the processes.

Note that I haven't used the other implementation which uses `MPI_Bcast` followed by `MPI_Reduce` to get the local sums. I couldn't find any proof as to why that method works, and if that's an approximation to the actual non-parallelized K-means algorithm or not.

Selecting K value

For choosing the value of K, I have firstly used my own files `extract_csv.sh` to convert the binary iles into csv data for easier and more pythonic processing. Now, I have used the Average Silhoutte Method and the Elbow method to plot graphs for various values of K. For the first dataset, the elbow method graph is rather smooth and hence it's completely upto us to choose the number of clusters. Hence, I've chosen K = 90 where we can see that the rate of change of derivative of curve is largest. The graph for the first dataset is shown below:

One major issue that I faced while plotting these graphs was the fact that the KMeans fitting step in sklearn is extremely slow for the given data1 or data2 sets. Hence, I had to plot these graphs over a span of ~20 hours. Furthermore,

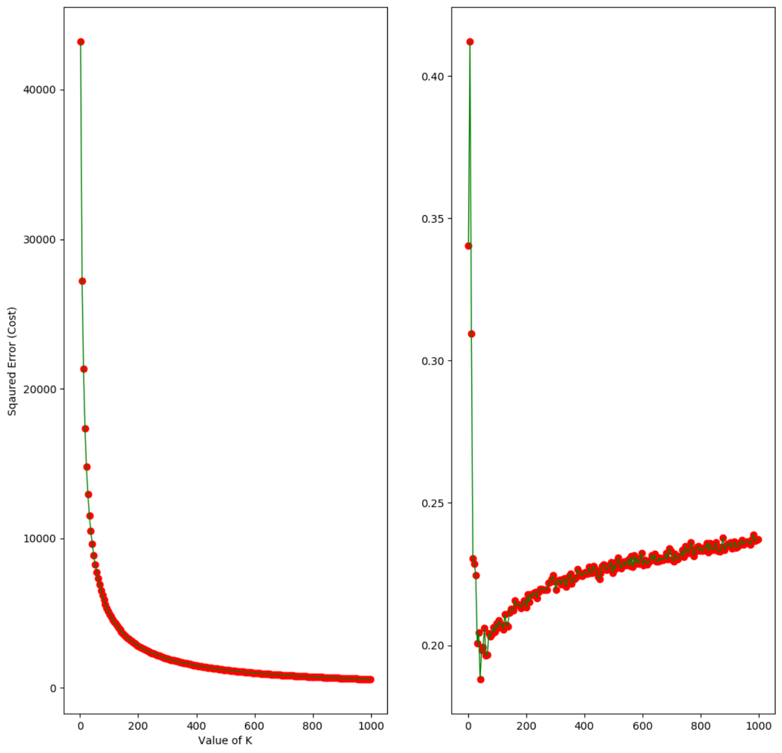

Figure 1: Elbow method and average silhoutte score vs the number of K

to speed up the process, I have used `MiniBatchKMeans` but even then it isn't fast enough so that we can compute the K value optimally for each time step file. As a result of time constraints, I have fixed K value. For the data2 dataset, the value using elbow method comes out to be around 10, and I have chosen 11 as K value for that.

How to run the code

** The `data1` and `data2` must be present in the same folder as these scripts are present. **

For the CSE cluster, running the `run.sh` script will run the code for all configurations and generate the output files in `cse/` sub-directory.

For the HPC cluster, running the `sub.sh` script will run the code for all configurations and generate the output files in `hpc/` sub-directory. ## Observations

1. Clustering is considered a difficult problem to parallelize since often, the global centroid updation step depends upon the local centroids calculated for each process.
2. From the plots I have stored in `Assignment3/cse/data*/plot.png` and `Assignment3/hpc/data1/plot.png`, we can see that we see 5-10 fold speed up using 8 processor. As we increase the number of processors, the speed up remains same owing to added network latencies and less network bandwidth compared to shared memory based IPC mechanisms when processors are running on same node.
3. The processing times show quite a bit of spread (reason may be different network bandwidths for different nodes), while the total time follows a predictable downward sloping curve.

Extra things which I tried out

I tried out Hierarchial Clustering using Octree (since it's a 3D problem) but there aren't any good resources available online for that. Also, the parallelizing the hierarchial clustering problem is much harder since there's inherent sequentiality meaning that previous results are needed for next level of processing.

I also looked at DBScan algorithm, but the parallel implementation is much more complex and probably beyond the scope of this assignment.