Efficient Parallel kNN Joins for Large Data in MapReduce

I. Summary

This paper proposed novel (exact and approximate) algorithms in MapReduce to perform efficient parallel kNN joins in large data, including BNLJ, its improved version using the R-tree indices, and a MapReduce-friendly, approximate algorithm based on z-values.

In H-BNLJ(Hadoop Block Nested Loop Join), each of R and S will be partitioned into n equal-sized blocks in the Map phase. Then, every possible pair of blocks will be sent into a bucket. After performing local kNN join in a bucket, the results will be aggregated to find the global kNNs for every record $r \in R$. H-BRJ(Hadoop Block R-tree Join) improve H-BNLJ by using R-tree to index each block of S.

However, algorithms above create excessive communication and computation costs. In order to achieve linear communication and computation cost, a approximate algorithm called H-zkNNJ(Hadoop based zkNN Join) was introduced.

In zkNN Join in MapReduce, a key issue is to determine what partition values, as $\{z_{i,1}, ..., z_{i,n-1}\}$, delivers good efficiency in a distributed and parallel computation environment like MapReduce. The paper presents a simple way to get a approximate equal sized partition with efficiency: sample each point in R and S with a probability, and determine the partition boundary based on this points.

As a result, there are 3 phases in zkNN Join:

- Phase 1 takes in R and S, then shifts them with vectors $\{v_0, ..., v_n\}$ and find the partition boundary for each R_i andS_i parallelly. The output of phase 1 is the approximate boundaries for R_i and S_i.
- Phase 2 takes in R_i and S_i, partition them with boundaries got in Phase 1, after shuffling and sorting, $R_{i,j}$ and $S_{i,j}$ will be sent to the same reducer for binary searching. The output of this phase is (rid,sid,d(r,s)) for each $s \in kNN(r, C_i(r))$.
- Phase 3 decides kNN(r,C(r)) for any $r \in R$ from the $kNN(r, C_i(r))'s$ emitted by the reducers at the end of phase 2.