Efficient Processing of k Nearest Neighbor Joins using MapReduce

I. Summary

The paper illustrates a efficient processing of kNN Joins using MapReduce

In the preliminaries, the authors shows the basic concepts of kNN Join, MapReduce Framework, and particularly the Voronoi Diagram-based Partitioning, which is the mathematic fundamental for partitioning the input set R and S.

A native and straightforward idea of performing kNN join in MapReduce is similar to the hash join algorithm. R is split into disjoint subsets, each subset R_i is distributed to a reducer, and without any pruning rule, the entire set S has to be sent to each reducer to be joined with R_i.

In order to reduce the shuffling cost, a better strategy is that R is partitioned into N disjoint subsets and for each subset R_i, find a subset of S_i that $R_i \ltimes S = R_i \ltimes S_i$ and $R \ltimes S = \bigcup_{1 \leq i \leq N} R_i \ltimes S_j$. So S_i is sent to the reducer that R_i belongs to and the KNN join is performed between R_i and S_i only.

For the purpose of reducing the size of S_i, we need to derive a distance bound based on the partitionning of R. Using basic geometry knowledge, the upper bound distance from $s \in P_j^S$ to $r \in P_i^R$, denoted as $ub(s, P_i^R)$, can be calculated easily. Then a bound(denoted as θ_i) of the KNN distance for all objects in P_i^R, can be touched by continually adding $ub(s, P_i^R)$, which $s \in KNN(p_j, P_j^S)$, to a priority queue for each P_j^S, until the size grows to k and there is no more $ub(s, P_i^R)$ smaller than the top of the queue.

The corresponding lower bound can be derived in a similar way, and after that, we get the necessary condition s is assigned to S_i:

$$|s, p_j| \geq LB(p_j^S, P_i^R) \tag{1}$$

So, given the input set R and S, the partition of R_i and S_i should be calculated first in the map procedure, and then each R_i,S_i pair will be sent to the same reducer to execute KNN join respectively. The details are shown below, and it takes three steps to complete the kNN Join.

- First, the master node invokes a preprocessing step, which takes the original R and S as input and finds out a set of pivot objects based on the input dataset R. The pivots can be got by Random Selection, Farthest Selection or k-means selection, and the pivots is used to create a Voronoi diagram, which can help partition objects in R effectively while preserving their proximity.
- Second, the first MapReduce job consists of a single Map phase, which takes the selected pivots and datasets R and S as the input. The output of the mapping phase is a partitioning on R, based on the Voronoi diagram of the pivots. Meanwhile, the mappers also collects some statistics about each partition R_i.
  The partitioning on R can be represented as a table, which including each object o along with its partition id, original dataset name (R or S), distance to the closest pivot.
  The statistics are kept in two summary table T_R and T_S. T_R maintains the following information for every partition of R: the partition id, the number of objects in the partition, the minimum distance L(P_i^R) and maximum distance U(P_i^R) from an object in partition P_i^R to the pivot. Moreover, T_S also maintains the distances between objects in KNN(p_i, P_i^S) and p_i
- Third, taking T_R, T_S and statistics from steps 2 as input, S_i will be built with the constraint of equation 1. The R_i and S_i will be sent to the same reducer to execute $R_i \ltimes S_i$. In the reducer, each value pair will be parsed to derive the partition P_i^R and subset S_i that consists of $P_{j_1}^S, ..., P_{j_M}^S$. then, a similar pruning process as the second step does, can be applied to S_i to get $LB(P_j^S, P_i^R)$ for every KNN distance, θ_i, for all objects of P_i^R. Hence, we can issue a range search with query and threshold θ_i over dataset S_i. After checking all partitions of S_i with pruning and updating KNN(r,S), the reducer outputs KNN(r,S).

The way S_i derived will bring replications of S. To minimize the number of replicas of objects in S, a intuitive way is to increase the number of pivots. However, this requires a large number of reducers, which may not be practical. A natural idea is to divide partitions of R into disjoint groups. There are two strategies for grouping.

- Geometric Grouping first select N p_i as the basic element of each group G_i, which is faraway from each other, and assign the rest p_j to the nearest group G_i.
- Greedy Grouping tries to minimize the increasing of $RP(S, G_i)$ when assigning a new partition P_j^R to G_i. But for the sake of reducing the computation cost, once $\exists s \in P - l^S$ satisfying the necessity of being added to S_i, we add all objects of partition P_l^S to $RP(S, G_i)$.

The draw back of the algorithm is that, improper pivot selection will cause large difference in partition size, which degrades performance due to unbalanced workload. And this paper doesn't present a pivot selection algorithm that will guarantee the uniformity of partitioning.