Similarity Group By for Big Data Analytics

Faculty: Yasin Silva  Students: Jeremy Starks, Emerson Cristal, Manuel Sandoval Madrigal, Diana Prado, Xavier Wallace

Arizona State University

Motivation

The Problem
- Analyzing massive amounts of data is critical for many commercial and scientific applications.
- However, this task can require processing tens to hundreds of terabytes of data.
- Big Data Systems like Apache Hadoop and Spark and their MapReduce (MR) programming frameworks enable analyzing very large datasets in a highly parallel and scalable way.
- Grouping operations are among the most useful operators for data processing and analysis.

General Considerations
1. A similarity group is defined as a set of points where each point is within epsilon of each other.
2. We propose MR-SGB, a MapReduce-based algorithm to efficiently identify similarity groups in large datasets.
3. Our algorithm is based on partitioning the data into smaller partitions. Each partitioning round uses a set of special points named pivots. Each data point will be associated with the group corresponding to its closest pivot.
4. Even though the algorithm processes the data in parallel over many nodes, it guarantees that each similarity group is generated only once.

Algorithm for K Pivots
1. Partition the data [Map]
   - Duplicate the points in overlapping areas (each base partition is extended by epsilon)
   - Structure of each record: (RecordID, RecordContent, AssignedPartition, BasePartition)
   - BasePartition: This is the ID of the pivot that is closest to the current record
   - AssignedPartition: This is the ID of the pivot associated to the current partition
2. For each partition P
   - cluster the points in P [Reduce]
     - For each partition, we know the value of |y| by looking at the AssignedPartition component of any record
     - Structure of each cluster C_i: (SetOfPoints, f_1, f_2, ..., f_n)
       - Observe that the array has k elements, where k is the number of pivots
       - f_i is a binary flag that is 1 if there is at least one record x in the cluster such that x.BasePartition = i, 0 otherwise
3. For each partition P, output the clusters (without duplicating clusters) [Reduce]
   - For each Cluster C_i in partition P:
     - minFlag = index of minimum value in C_i.f_1, f_2, ..., f_n that is 1
     - If (i = minFlag) then output C_i, otherwise don’t output it (it will be outputted somewhere else)

Example: Case of 2 Pivots

Partitioning and Generation of Similarity Groups

Goals
- Partition the initial dataset into two partitions such that we can still identify all the similarity groups (G1-G3)
- Each similarity group should be generated in only one partition

Solution (using two pivots/partitions):
- Partition the input using two pivots (P_0 and P_1) such that each point belongs to the partition of its closest pivot
- Additionally, duplicate the points in the ε-neighborhood of these pivot points
- Identify the similarity groups in each partition as follows:
  - In partition P_0:
    - If group
      - Solely in A Generate
      - In A and C Generate
      - Solely in C Generate
      - In A and C and D Generate
      - In C and D Generate
      - In D and B Generate
  - In partition P_1:
    - If group
      - Solely in B Generate
      - In A and C Generate
      - Solely in C Generate
      - In A and C and D Generate
      - In C and D Generate
      - In D and B Generate

Example: Initial Dataset (2D space)

Dataset
1. Real dataset:
   - Source: YearPredictionMSD (UCI ML Repository)
   - Data type: numeric vector data (90D)
   - Size (Scale Factor 1): 200K records
2. Synthetic dataset
   - Our generator enables the customization of:
     - # of records per group and record repetition
     - # of Scale Factors (SF) and # of records per SF
     - Epsilon value
     - Dimensionality
   - Format: Line ID, Aggregation Value, Vector
   - Size (Scale Factor 1): 200K records

Experiments
1. Execution time varying dataset size (SF1-SF5)
2. Execution time varying dimensionality (1D, 2D, 100D, 200D)
3. Execution time varying epsilon (1%-5%)

Algorithms
1. Implemented using Hadoop and MapReduce
2. Similarity Group-by (MR-SGB): proposed similarity grouping operator

Future Work
1. Implement Group-By, K-Means, and Similarity Group-By in Apache Spark to compare the algorithms across the two popular distributed computing frameworks.
2. Conduct thorough experimental evaluation using the generated real and synthetic datasets across the two frameworks (Hadoop and Spark).
3. Prepare a publication detailing the design, implementation details and performance comparison of Similarity Group-By and alternative algorithms.

References

Sample Diagram:
- Diagram showing the partitioning and generation of similarity groups in a 2D space with two pivots (P_0 and P_1).