MapReduce-based Similarity Grouping for Big Data

Faculty: Yasin Silva Students: Adelbert Wadsworth, Kyle Brown, Jason Reed (alumnus)

Arizona State University

Motivation

The Problem

- The analysis of massive amounts of data is a crucial requirement in many commercial and scientific applications.
- Analyzing these data sets may require processing tens or hundreds of terabytes of data.
- Big Data Management Systems constitute an answer to the requirements of processing massive datasets in a highly scalable and distributed fashion.
- Grouping operations, which aggregate large datasets and compute aggregation functions for each group, are considered among the most useful operations for data processing and analysis.
- Similarity groups contain similar records instead of exact matches, and enable the composition of more useful data analysis queries.

Our Contribution

- Design of a highly distributed and scalable algorithm (MR-SGB) that performs similarity grouping on massive datasets.
- Implementation of the proposed algorithm on Apache Hadoop, a widely used open source Big Data system.

General Considerations

1. A similarity group is defined as a set of points where each point is within epsilon from any other point in the group.
2. MR-SGB is proposed as:
   1. An initial way to explore the distribution and clusters in the data.
   2. An alternative and more compact way to identify pairs of similar objects (Similarity Join).
3. Our algorithm will be based on partitioning the data into smaller partitions. The algorithm will partition the data around a set of special points named pivots.
4. We will have one partition for each pivot.
5. A cluster could potentially be contained in multiple partitions but it should be outputted/generated in only one.
6. Our criteria will be to output the cluster in the partition that corresponds to the minimum base partition of the points contained in the cluster.
   For instance if we have 4 pivots and the cluster has points that belong to base partitions 2 and 3, then this cluster should be outputted when Partition 2 is being processed.

General Algorithm (Two Rounds)

Key properties of the single-round algorithm:

- We can increase the number of pivots (*k*) such that all the partitions are small enough to be processed in a single node
- For the unlikely case that we still have a large partition, we will add a second level of partitioning

Algorithm:

- Execute algorithm (round 1)
- For each partition P_i
  - If P_i can be processed in a single node, we do this
  - Else, we save P_i in the distributed file system
- For each partition P_i that was saved
  - Execute second-level algorithm (round 2)

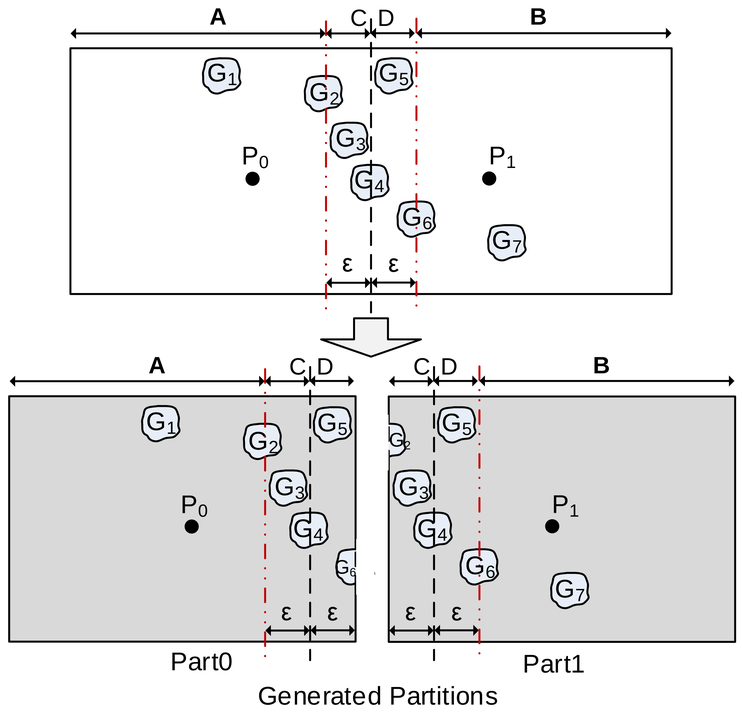
The second level algorithm is similar to the first level one with the difference that we need to keep track of the history of flags for each partition.

Algorithm for k Pivots (First Round)

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_i, **cluster** the points in P_i [Reduce]
   - For each partition, we know the value of *i* by looking at the *AssignedPartition* component of any record
   - Structure of each cluster C_n: {SetOfPoints,[f_1, f_2, …, f_k]}
     - Observe that the array has *k* elements, where *k* is the number of *pivots*
     - f_s is a binary flag that is 1 if there is at least one record *X* in the Cluster such that *X.BasePartition* = *s*, 0 otherwise
3. For each partition P_i, **output** the clusters (without duplicating clusters) [Reduce]
   - For each Cluster C_n in partition P_i
     - *minFlag* = index of minimum value in C_n.[f_1, f_2, …, f_k] that is 1
     - If (*i* = *minFlag*) then output C_n, otherwise don't output it (it will be outputted somewhere else)

Example: Case of 2 Pivots

Initial Dataset (2D space)



Partitioning and Generation of Similarity Groups

Goals:

- Partition the initial dataset into two partitions such that we can still identify all the similarity groups (G_1-G_7)
- 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-windows (C and D). Part0 = A+C+D, Part1 = C+D+B.
- Identify the similarity groups in each partition as follows:

In partition Part0:		**In partition Part1:**	
If group	Then	If group	Then
Solely in A	Generate	Solely in C	Ignore
In A and C	Generate	In C and D	Ignore
Solely in C	Generate	Solely in D	Generate
In C and D	Generate	In D and B	Generate
Solely in D	Ignore	Solely in B	Generate

- In the example, similarity groups G_1, G_2, G_3, and G_4 are generated in Part0 while G_5,G_6, and G_7 in Part1