Similarity Grouping for Big Data

Faculty: Yasin Silva Students: Nathan Middlebrook and Jeremy Starks

Arizona State University

Motivation

The Problem:

- Analyzing massive amounts of data is now critical for many commercial and scientific applications.
- The analysis of such datasets may require processing tens, possibly hundreds of terabytes of data.
- Big Data Management Systems comprise a solution to the requirements of processing massive datasets in a highly scalable and distributed fashion.
- Grouping operations (large dataset aggregators, with group-specific functions) are considered among the most useful operations for data processing and analysis.
- Similarity groups contain similar records, instead of exact matches, thus enabling the composition of more useful data analysis queries.

Our Contribution:

- The design of a highly distributed, 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 framework.

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:
   A. An initial way to explore the distribution and clusters in the data.
   B. An alternative, 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 generate these partitions 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; however, it should be output into/generated in only one of them.
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: given 4 pivots and the cluster has points belonging to base partitions #2 and #3, then this cluster should be output when Partition #2 is being processed.

General Algorithm

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:

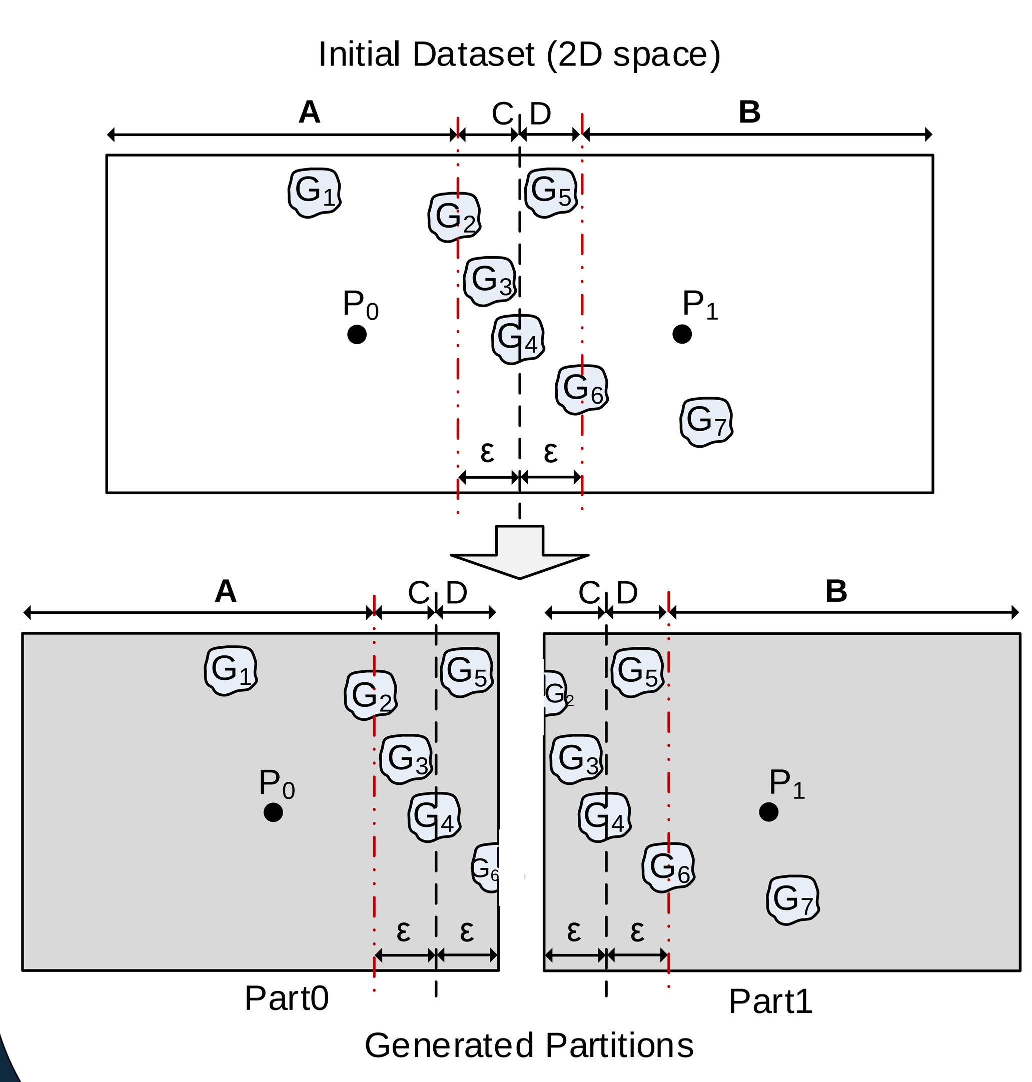
- Execute algorithm (round 1)
- For each partition P_i
  - If P_i can be processed in a single node, then we do so
  - Else, we save P_i in the distributed file system
- For each partition P_i that was saved
  - Execute the second-level algorithm (round 2)

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

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_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



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:

If group	Then
Solely in A	Generate
In A and C	Generate
Solely in C	Generate
In C and D	Generate
Solely in D	Ignore

In partition Part1:

If group	Then
Solely in C	Ignore
In C and D	Ignore
Solely in D	Generate
In D and B	Generate
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

Experimental Setup

Datasets

1. Real dataset: YearPredictionMSD (UCI ML Repository)
   Vector, 90D, size(SF1): 200K
2. Synthetic dataset
   Vector, 9xD, size(SF1): 200K

Experiments

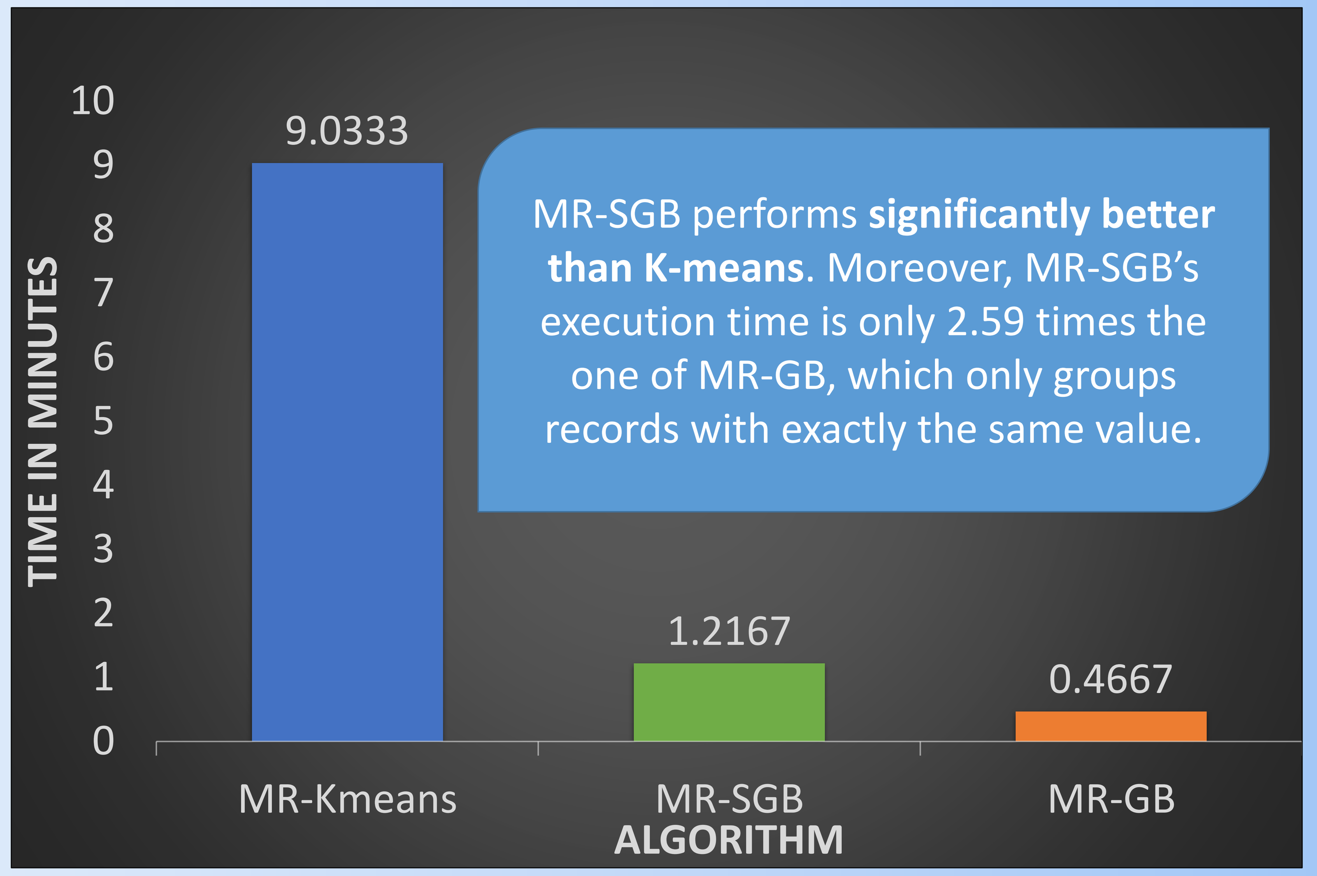
1. Execution time varying dataset size (SF1-SF5)
2. Execution time varying dimensionality (2D, 5D, 25D, 100D, 200D)
3. Execution time varying epsilon (1%-5%)

Algorithms (implemented using the MapReduce Big Data framework)

1. Similarity Group-by (MR-SGB): proposed similarity grouping operator
2. K-means (MR-Kmeans): standard clustering algorithm
3. Group-by (MR-GB): standard non-similarity-based database grouping operator

Preliminary Results

K-means (k=10), SGB (eps=1, pivots=10), GB (count), dataset (2D, small)



References

1) Silva, Y. N., Aref, W., Ali, M. Similarity Group-by. In: ICDE (2009)
2) Silva, Y.N., Reed, J.M.: Exploiting MapReduce-based similarity joins. In: SIGMOD (2012)
3) Silva, Y.N., Reed, J.M., Tsosie, L.M.: MapReduce-based similarity join for metric spaces. In: VLDB/Cloud-I (2012)
4) Vernica, R., Carey, M.J., Li, C.: Efficient parallel set-similarity joins using MapReduce. In: SIGMOD 2010 (2010)
5) Afrati, F.N., Sarma, A.D., Menestrina, D., Parameswaran, A., Ullman, J.D.: Fuzzy joins using MapReduce. In: ICDE (2012)
6) Okcan, A., Riedewald, M.: Processing theta-joins using MapReduce. In: SIGMOD (2011)
7) Metwally, A., Faloutsos, C.: V-SMART-join: a scalable MapReduce framework for all-pair similarity joins of multisets and vectors. In: VLDB (2012)
8) Xiao, C., Wang, W., Lin, X., Yu, J.X.: Efficient similarity joins for near duplicate detection. In: WWW (2008)