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 framework 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_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)

Test Setup

Datasets

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

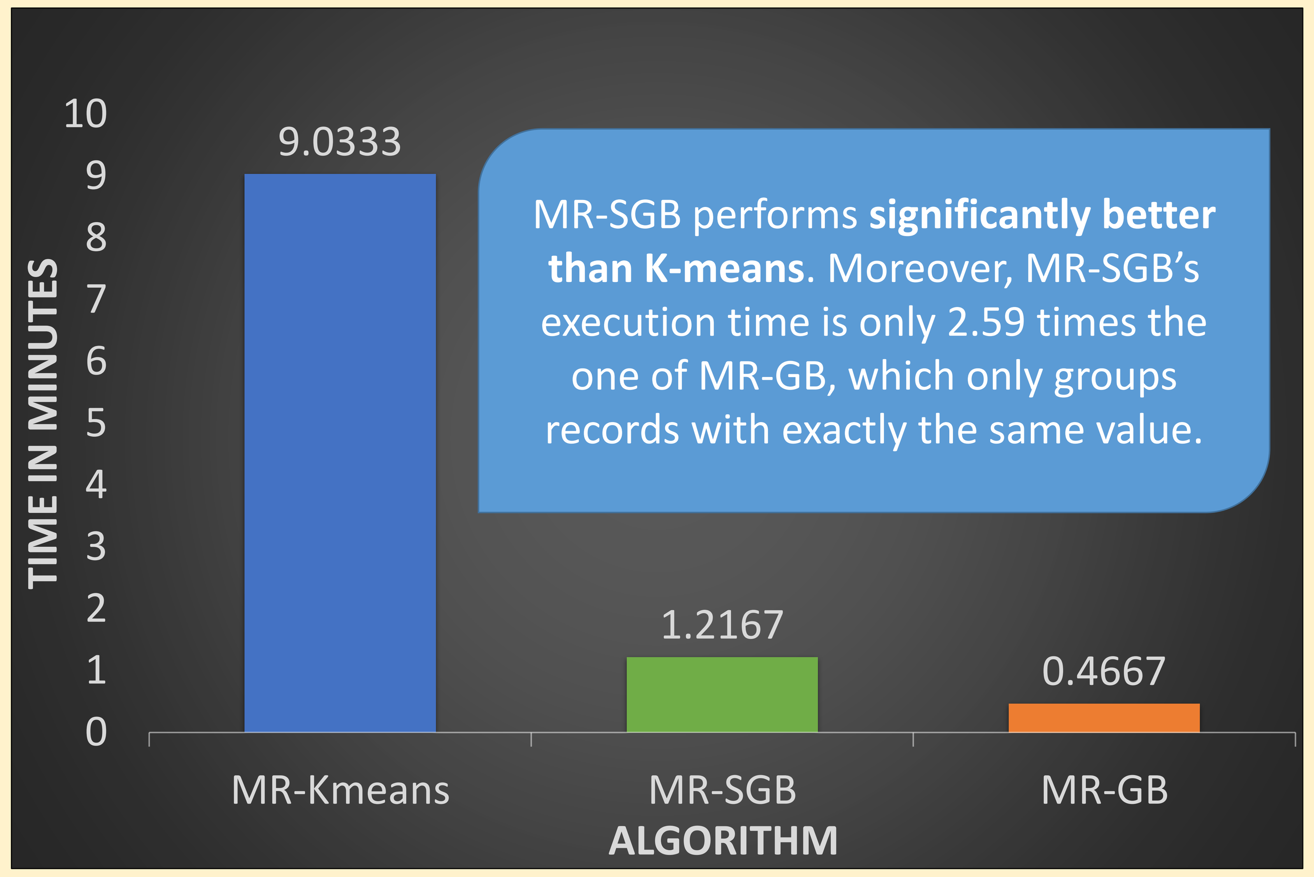
1. Execution time varying dataset size (SF1-SF5)
2. Execution time varying dimensionality (10D, 25D, 100D, 200D)
3. Execution time varying epsilon (1%-5%)

Algorithms

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

Preliminary Results

- Implemented and compared implemented algorithms using Apache Hadoop
- Compared algorithms: MR-GB, MR-Kmeans, and MR-SGB
- All the algorithms were modified to accept the same input data format



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

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 support additional partitioning rounds.

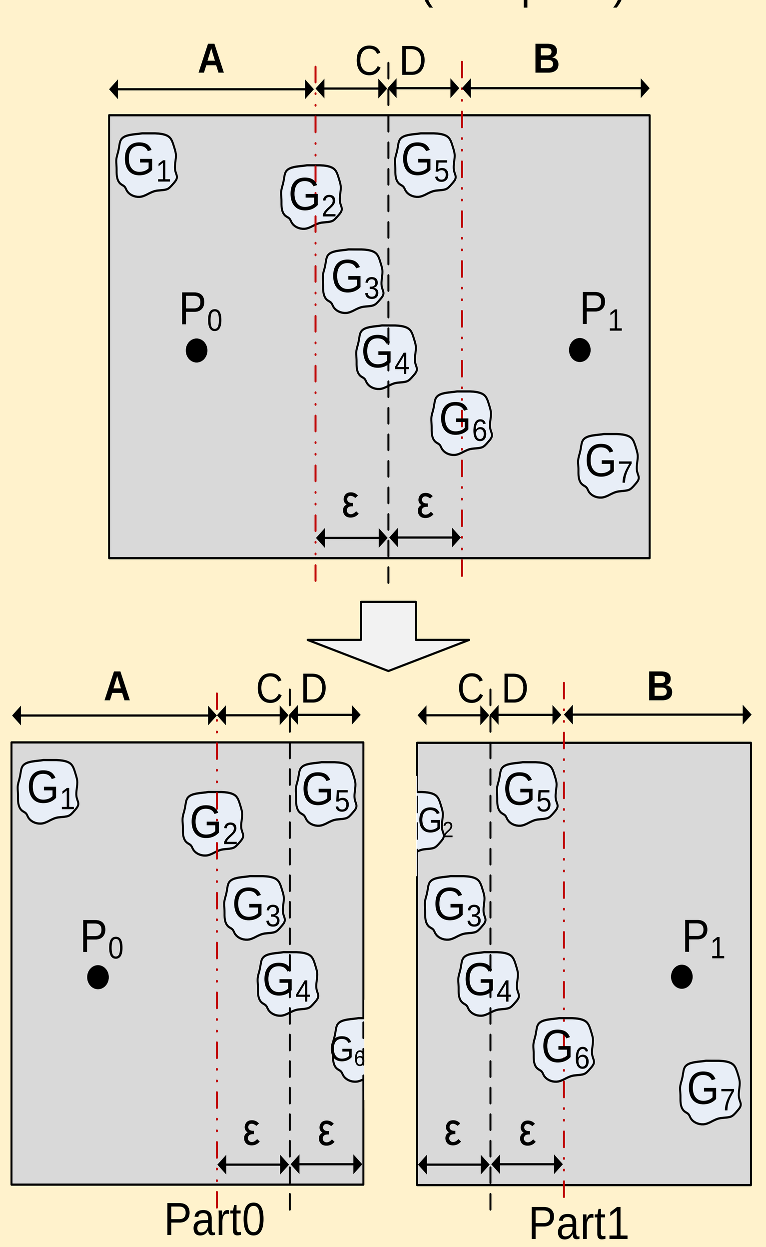
Overall Algorithm:

- 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 (DFS)
- For each partition P_i saved into the DFS
  - Execute a new round to re-partition P_i

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

Example: Case of 2 Pivots

Initial Dataset (2D space)



Generated Partitions

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

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

1) Tang, M., Tahboub, R., Aref, W., Atallah, M., Malluhi, Q., Ouzzani, M., Silva, Y. N. *Similarity Group-by Operators for Multi-dimensional Relational Data*. IEEE Transactions on Knowledge and Data Engineering (TKDE), 28, 2, pp 510-523, 2016.
2) Silva, Y. N., Aref, W., Ali. M. *Similarity Group-by*. In: ICDE (2009)
3) Silva, Y.N., Reed, J.M. *Exploiting MapReduce-based similarity joins*. In: SIGMOD (2012)
4) Silva, Y.N., Reed, J.M., Tsosie, L.M. *MapReduce-based similarity join for metric spaces*. In: VLDB/Cloud-I (2012)
5) Vernica, R., Carey, M.J., Li, C. *Efficient parallel set-similarity joins using MapReduce*. In: SIGMOD 2010 (2010)
6) Afrati, F.N., Sarma, A.D., Menestrina, D., Parameswaran, A., Ullman, J.D. *Fuzzy joins using MapReduce*. In: ICDE (2012)
7) Metwally, A., Faloutsos, C. *V-SMART-join: a scalable MapReduce framework for all-pair similarity joins of multisets and vectors*. In: VLDB (2012)