Similarity Grouping for Big Data – Experimental Evaluation

Faculty: Yasin Silva Students: 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 Apache Spark and their programming frameworks enable the analysis of very large datasets in a highly parallel and scalable way.
- Grouping operations are among the most useful operators for data processing and analysis.
- Simple grouping operations are fast but don't capture complex groups. Clustering techniques capture complex groups but are slow.

Our Solution

1. A *similarity group* is defined as a set of points where each point is within **epsilon** of each other.
2. We propose **MRSGB** and **SPSGB**, a both MapReduce and Spark 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.

Test Setup

Dataset Generator

- Our parametrized data generator produces datasets that contain clusters with certain properties.
- The generator enables the customization of:
  - Number of Groups
  - # of records per group and record repetition
  - # of Scale Factors (SF) and # of records per SF
  - Epsilon value
  - Dimensionality
- Record format: Line ID, Aggregation Value, Vector
- Size: 200K records (Scale Factor 1) - 1M records (SF5)

Experiments

1. Execution time varying dataset size (SF1-SF5)
2. Execution time varying dimensionality (200D, 300D, 400D, 500D)
3. Execution time varying number of nodes (2, 4, 6, 8, 10 nodes)

Platform

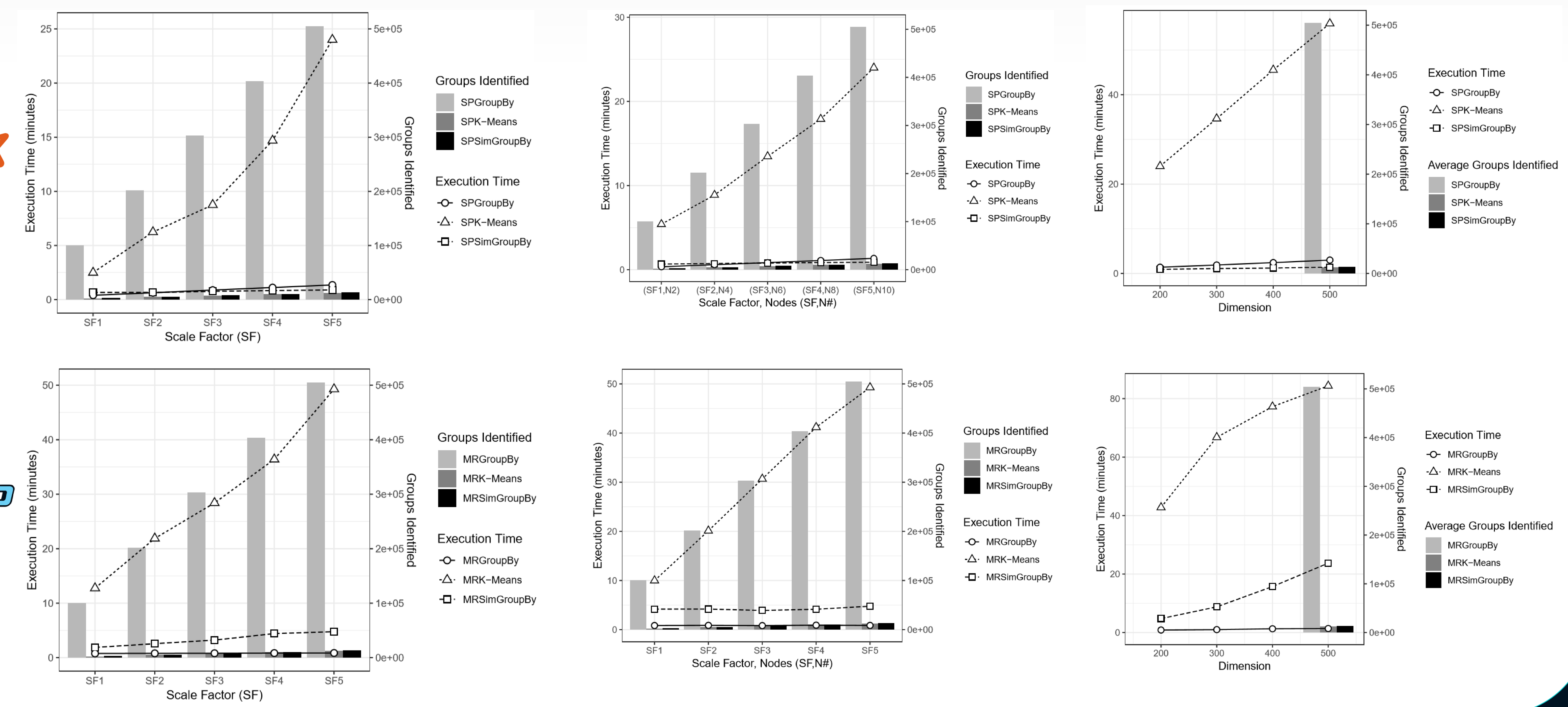
- Cloud-based computer clusters in **Google Cloud Platform**

Algorithms

1. Implemented using Hadoop (MapReduce) and Spark
2. **Similarity Group-by** (MRSimGroupBy, SPSimGroupBy): proposed similarity grouping operator
3. **K-means** (MRK-Means, SPK-Means): standard clustering algorithm
4. **Group-by** (MRGroupBy, SPGroupBy): standard non-similarity-based database grouping operator

Experimental Results

Performance with Increasing Dataset Size | Performance while Increasing Cluster Size and Dataset Size | Performance with Increasing Dimensionality



General Similarity Grouping Algorithm

Overall Algorithm:

- Execute the next round
- For each partition P_i obtained in this round
  - If P_i can be processed in a single node, then we do so
  - Else, we save P_i for further processing
- For each partition P_i saved for further processing
  - Execute a new round to re-partition P_i

Key Properties of Each Round of the Algorithm:

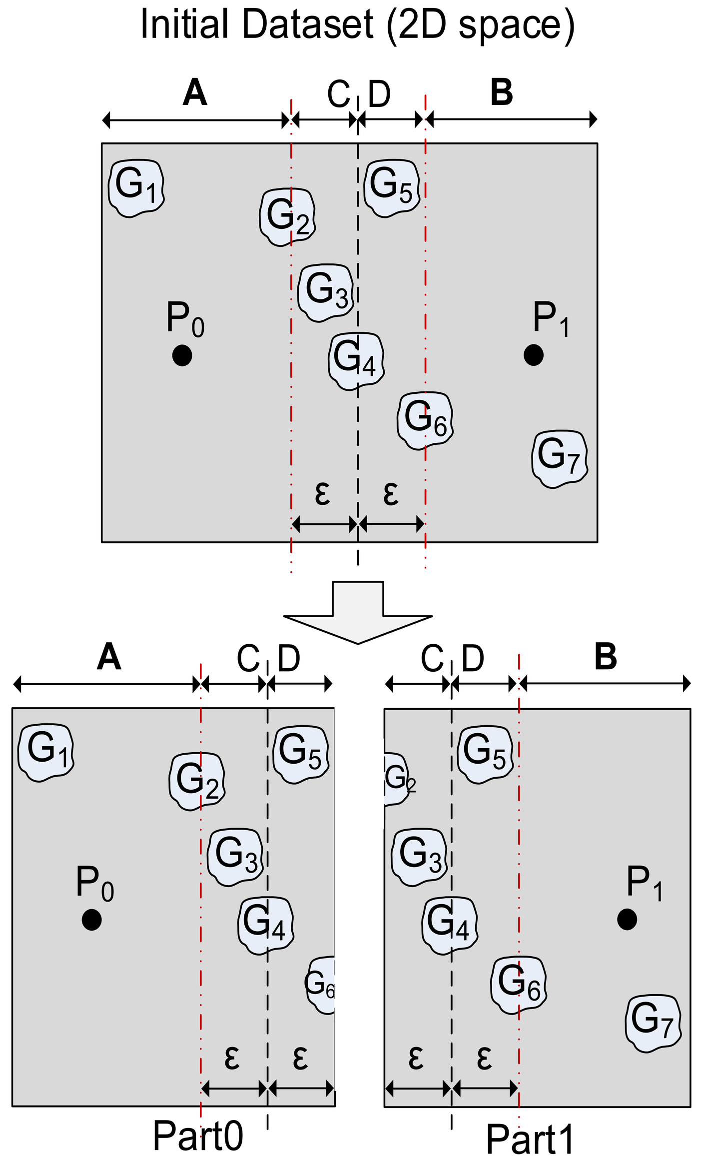
- We can increase the number of pivots (k) such that all the partitions are small enough to be processed in a single node.
- For the unlikely case that we still have a large partition, we support additional partitioning rounds.

To properly support a multi-round approach that only outputs each identified cluster once, we keep track of the history of partitions that a records has been assigned to during the execution of our algorithm.

Algorithm of a Single 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



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

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)

www.public.asu.edu/~ynsilva/SimCloud/