Similarity Grouping in Big Data Systems

Yasin N. Silva, Manuel Sandoval, Diana Prado, Xavier Wallace, Chuitian Rong
Arizona State University

Motivation

The Problem

- Analyzing massive amounts of data is critical for many commercial and scientific applications.
- Big Data Systems like Apache Hadoop and Spark 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 are limited to equality-based grouping. More sophisticated grouping techniques capture complex groups but often at a steep increase in execution time.
- Previous work introduced the Similarity Grouping (SG) operator which aims to have fast execution times and capture complex groups. SG, however, was proposed for single node relational database systems.

Our Contributions

1. We introduce the Distributed Similarity Grouping (**DSG**) operator to efficiently identify similarity groups in big datasets.
2. DSG supports the identification of similarity groups where all the elements of a group are within a given threshold (Ɛ) from each other.
3. DSG guarantees that each group is generated only once.
4. DSG can be used with any metric and supports many data types.
5. We present guidelines to implement DSG in both Apache Spark and Hadoop.
6. We extensively assess DSG's performance and scalability properties.

Test Setup

Algorithms (Implemented using Apache Hadoop and Spark)

1. **Distributed Similarity Grouping (DSG)**: proposed similarity grouping operator
2. **K-means**: standard clustering algorithm
3. **Standard Grouping**: standard non-similarity-based grouping operator

Computer Cluster

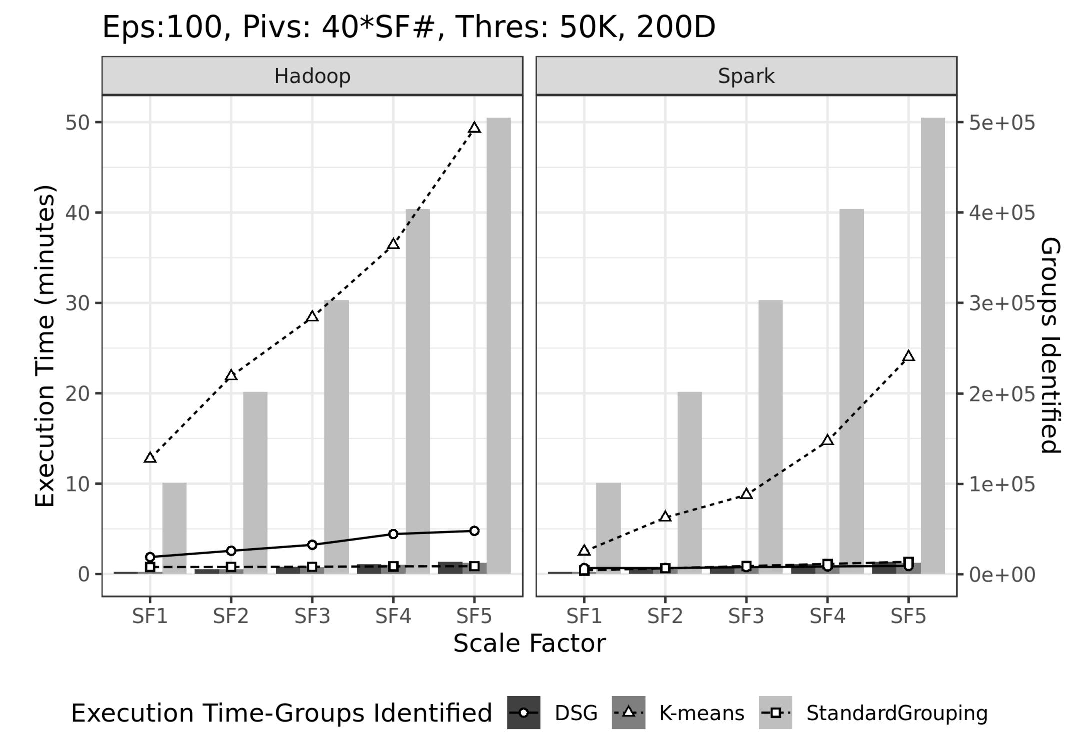
- Fully distributed clusters in **Google Cloud Platform**.
- Default cluster configuration:
  - One master
  - Ten worker nodes
- Each node used the Cloud Dataproc 1.3 image and had 4 virtual CPUs, 15 GB of memory and 500 GB of disk space.
- Number of reducers per Hadoop job: 0.95 × (# of worker nodes) × (# of vCPUs per node - 1)
- Number of splits per Spark job: 2 × (# of worker nodes) × (# of vCPUs)

Data

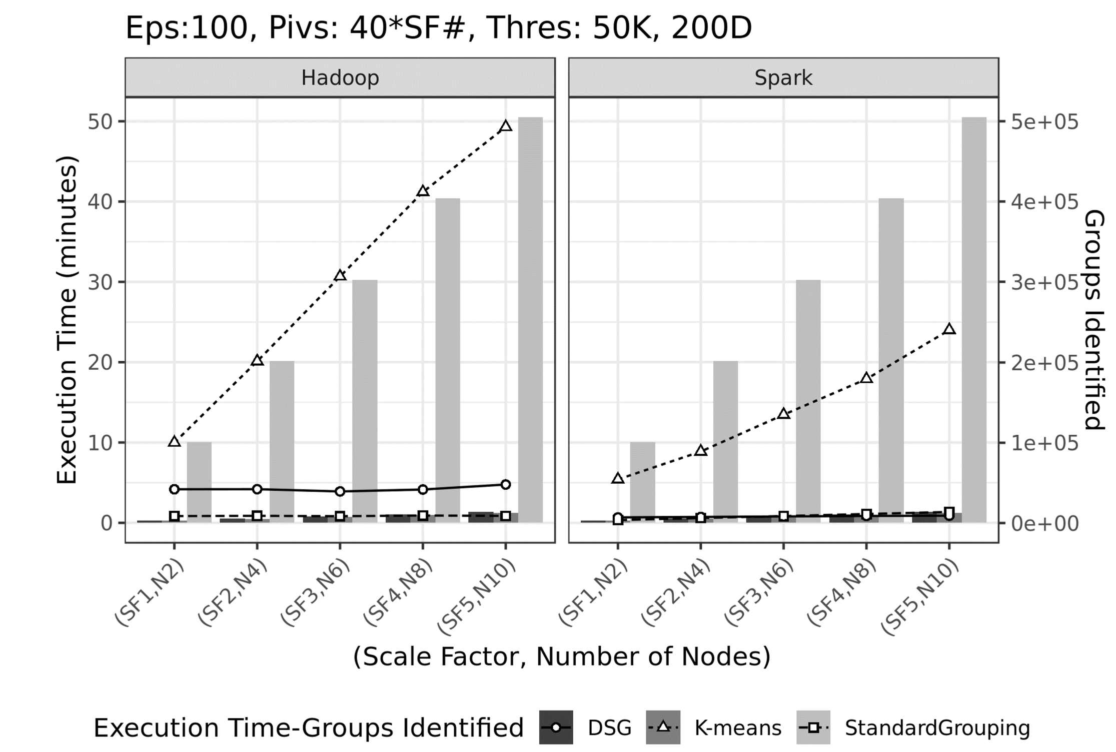
- We implemented a parametrized synthetic dataset generator.
- The datasets are composed of multidimensional vector-based similarity groups separated by 2Ɛ.
  - DSG and K-Means are expected to have the same output.
  - Standard Grouping only identifies equality-based groups.
- Each data record consisted of an ID, an aggregation attribute, and a multidimensional vector.
- Dataset Size (Scale Factor): 200,000 (SF1) – 1,000,000 (SF5)
- Dimensionality: 100D, 200D, 300D, 400D, and 500D
- The SF1 datasets contains about 13,000 similarity groups and each of them contained 50 to 100 records. Each record was duplicated between 1 and 3 times.

Experimental Results

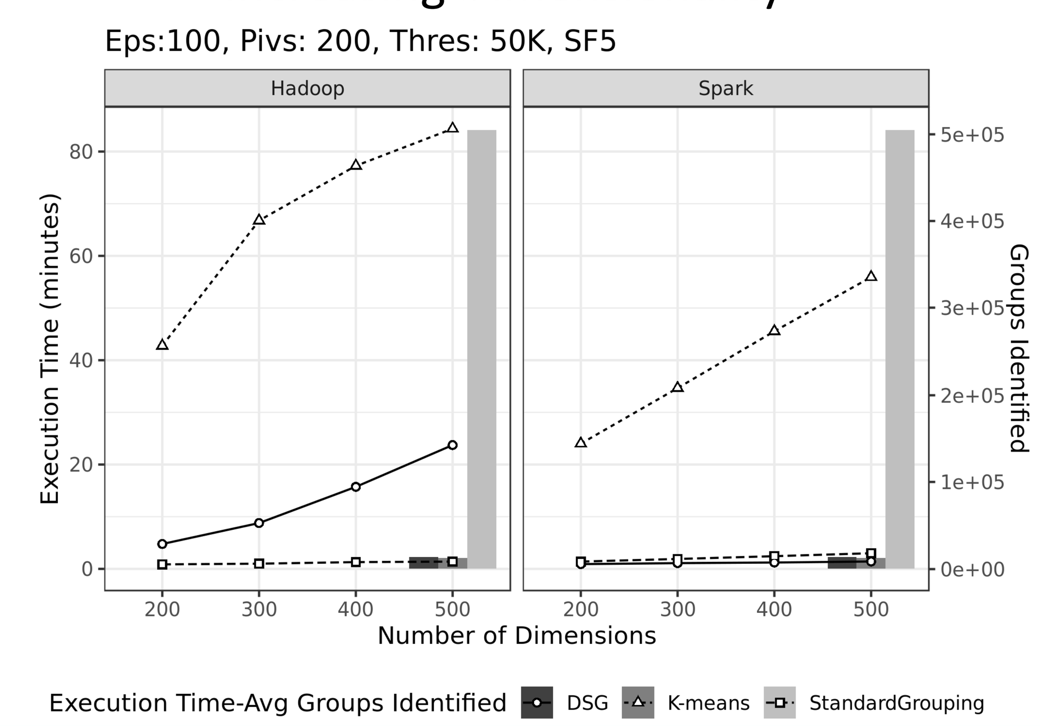
Increasing Dataset Size



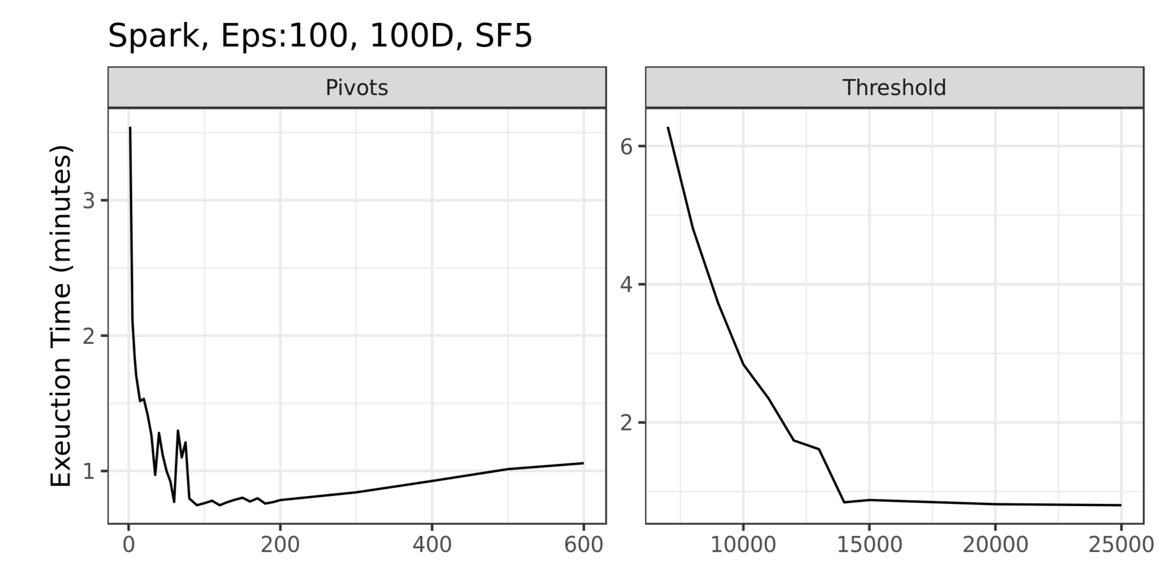
Increasing Dataset Size and Cluster Size



Increasing Dimensionality



Increasing Number of Pivots and Memory Threshold



General DSG Algorithm

- DSG uses pivot-based data partitioning to distribute and parallelize the computational tasks.
- The goal is to divide a large dataset into partitions that can be processed independently and in parallel to identify the similarity groups.
- The pivots are a subset of input data records and each pivot is associated with a partition.
- Each input record is assigned to the partition associated with its closes pivot. DSG also replicates the records at the boundary between partitions.
- If a partition is small enough to be processed at a single node, the algorithm will identify groups in that partition.
- If this is not the case, the partition is stored for further processing in a subsequent round
- DSG is a multi-round algorithm.
- In practice, we can increase the number of pivots such that all the partitions are small enough to be processed in a single round.
- DSG keeps track of the history of partitions assigned to each record.

Overall Algorithm

- Partition the input data using a set of pivots
- 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 P_i saved for further processing
  - Execute a new round to re-partition P_i

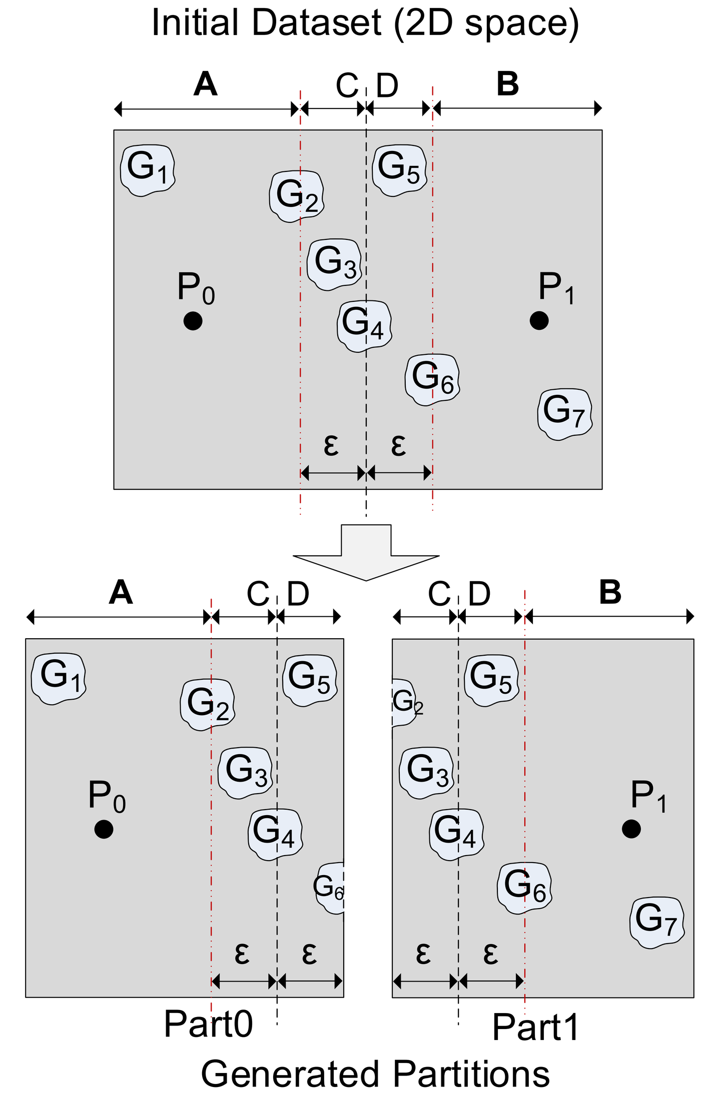
Main Algorithm

Algorithm 1 ***DistSimGrouping***
Input: *inputData*, *eps*, *numPivots*, *memT* **Output**: similarity groups in *inputData*

```
pivots = selectPivots(numPivots, inputData)
//Partitioning - r: ⟨ID, value, assignedPartitionSeq,
basePartitionSeq⟩
for each record r in a chunk of inputData do
  P_c = getClosestPivot(r, pivots)
  output ⟨P_c, r⟩ //intermediate output
  for each pivot p in {pivots-P_c} do
    if (dist(r, p) - dist(r, P_c))/2 ≤ eps then
      output ⟨p, r⟩ //intermediate output
    end if
  end for
end for
//Shuffle: records with same key => partition
//Group Formation
for each partition P_i do
  if size of P_i > memT then
    store P_i for processing in subsequent round
  else
    C_i = findSimGroups(P_i, eps) //C_i:{C_i_k},
    //C_i_k:⟨records, flags⟩, flags:{F_m}, F_m:{f_m_n}
    //Output Generation (without duplication)
    for each cluster C_i_k in partition P_i do
      generate minFlags //minFlags[o]={index
      //of 1st element in C_i.flags[o] equal to 1}
      aPartitionSeq = r.assignedPartitionSeq
      //r is any record in P_i
      if ∀o,minFlags[o]=aPartitionSeq[o] then
        output C_i_k //final output
      end if
    end for
  end if
end for
```

Example with Two 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

References

1) Apache. Hadoop. https://hadoop.apache.org/.
2) Apache. Spark. https://spark.apache.org/.
3) J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In OSDI, 2004.
4) F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. ACM Trans. Comput. Syst., 26(2): 1–26, 2008.
5) H. Garcia-Molina, J. Ullman, and J. Widom. Database Systems: The Complete Book. Pear-son, 2nd Edition.
6) J. Gray, A. Bosworth, A. Layman, and H. Pirahesh: Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. In ICDE, 1996.
7) S. P. Lloyd. (1982). Least squares quantization in PCM. IEEE Trans. on Information Theo-ry. 28 (2): 129–137, 1982
8) M. Ester, H.P. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovering Clusters. In KDD, 1996.
9) Y. N. Silva, W. G. Aref, and M. Ali. Similarity Group-by. In ICDE, 2009.
10) M. Tang, R. Y. Tahboub, W. G. Aref, M. J. Atallah, Q. M. Malluhi, M. Ouzzani, and Y. N. Silva. Similarity Group-by Operators for Multi-dimensional Relational Data. IEEE Trans. on Knowledge and Data Engineering, 28(2): 510-523, 2016.
11) P. Berkhin. Survey of clustering data mining techniques. Accrue Software, 2002.
12) M. Li, G. Holmes, and B. Pfahringer. Clustering large datasets using Cobweb and K-Means in tandem. The Australian Joint Conference on Artificial Intelligence, 2004.
13) F. Farnstrom, J. Lewis., and C. Elkan: Scalability for clustering algorithms revisited. SIGKDD Explorations Newsletter, 2 (1): 51–57, 2000.
14) S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large data-bases. In SIGMOD Record, 27(2): 73–84, 1999.
15) P. P. Anchalia, A. K. Koundinya and S. N. K. MapReduce Design of K-Means Clustering Algorithm. In ICISA, 2013.
16) Apache. Spark Clustering. https://spark.apache.org/docs/latest/ml-clustering.html.
17) Y. N. Silva, M. Arshad, and W. G. Aref. Exploiting Similarity-aware Grouping in Decision Support Systems. In EDBT, 2009.
18) E. H. Jacox and H. Samet. Metric space similarity joins. ACM Trans. Database Syst., 33(2):7:1–7:38, 2008.