Postprint: Spark-Based Parallel Density Peak Clustering Algorithm

Authors: Sun Weipeng, Wu Xisheng, Meng Bin

Date: 2018-10-11T00:00:00+00:00

Abstract

To address the problem of high overall time complexity in the FSDP clustering algorithm, which arises from the need to traverse the entire dataset when computing the local density and minimum distance of data objects, this paper proposes a Spark-based parallel FSDP clustering algorithm called SFSDP. First, the algorithm partitions the dataset to be clustered into multiple data partitions with relatively balanced data volumes through spatial grid partitioning. Then, it performs parallel clustering analysis on the data within each partition using an improved FSDP clustering algorithm. Finally, it generates a global cluster set by merging the local clusters across partitions. Experimental results show that SFSDP can effectively conduct clustering analysis on large-scale datasets and demonstrates excellent performance in both accuracy and scalability compared to the FSDP algorithm.

Full Text

Preamble

Title: Spark-Based Parallel Density Peak Clustering Algorithm

Authors: Sun Weipeng[1], Wu Xisheng[1], Meng Bin[2]
[1]School of IoT Engineering, Jiangnan University, Wuxi, Jiangsu 214122, China
[2]Software Engineering Center, China Shipbuilding Group No. 702 Institute, Wuxi, Jiangsu 214082, China

Abstract: The FSDP clustering algorithm suffers from high overall time complexity because it must traverse the entire dataset when calculating the local density and minimum distance of data objects. To address this problem, this paper proposes a Spark-based parallel FSDP clustering algorithm called SFSDP. First, the algorithm divides the dataset into multiple partitions with relatively balanced data volumes through spatial grid partitioning. Then, it performs clustering analysis on the data within each partition in parallel using an improved

FSDP algorithm. Finally, it merges local clusters from different partitions to generate global clusters. Experimental results demonstrate that SFSDP can effectively perform clustering analysis on large-scale datasets compared with FSDP, exhibiting good performance in both accuracy and scalability.

Keywords: clustering; density peak; space division; parallel; Spark

0 Introduction

With the rapid global proliferation of the Internet, people face massive amounts of data daily from social, commercial, medical, engineering, scientific, and everyday life domains. This data explosion, widespread availability, and enormous scale have ushered us into a true data era. The need to quickly and conveniently extract useful information from these unstructured, chaotic large-scale datasets and transform them into knowledge has given birth to data mining [1]. In the field of data mining, cluster analysis serves as a common data analysis method that can partition a collection of data objects into multiple clusters without any prior knowledge, ensuring that objects within the same cluster are similar to each other but dissimilar to objects in other clusters [2]. To date, numerous researchers have proposed various clustering algorithms for different application scenarios [3], significantly advancing cluster analysis research. In real life, cluster analysis has been widely applied in many fields, including web search, decision analysis, image pattern recognition, automatic document summarization, natural language processing, business intelligence, biology, and more.

FSDP (Clustering by Fast Search and Find of Density Peaks) represents a novel density-based clustering algorithm [4] that identifies cluster centers as data objects with relatively high local density and large distances from other high-density objects. Each non-center data object is then iteratively assigned to the cluster of its nearest neighbor with higher density along the direction of increasing density. Since its proposal, FSDP has attracted considerable attention and research from scholars [5–8]. Although FSDP offers advantages such as simple principles and the ability to discover clusters of arbitrary shapes and sizes, its efficiency remains relatively low when handling large-scale or high-dimensional clustering tasks. This limitation arises because FSDP requires high time complexity to compute the local density and minimum distance for each data object in the dataset, making it unsuitable for large-scale data clustering.

To address the efficiency and scalability issues of FSDP when clustering massive, high-dimensional data, this paper proposes a Spark-based parallel FSDP clustering algorithm called SFSDP. The algorithm first partitions massive datasets into multiple balanced data partitions in space, then distributes these partitions to computing nodes in a cluster for independent clustering, and finally merges local clustering results to generate global clusters. Experimental results show that SFSDP, compared with FSDP, maintains accuracy while demonstrating strong scalability and effective large-scale dataset clustering capabilities.

2 Problem Description and Related Definitions

Problem Description: Given a dataset to be clustered, its d-dimensional spatial region DS is called the data space of dataset D, where each data object represents a point in d-dimensional space and denotes the projection of the data object onto the k-th dimension axis. The SFSDP clustering algorithm can obtain a set of clusters from this dataset.

The SFSDP clustering algorithm is based on the following definitions:

Definition 1 (Spatial Grid and Grid Cell): The data space DS is divided into multiple non-overlapping regions. This grid division is called a spatial grid, denoted as G. Each subregion is a grid cell of the spatial grid, denoted as g. [Figure 1: see original paper] illustrates the partitioning of a two-dimensional data space DS, where the large rectangle bordered by solid lines represents the data space DS, and the smaller rectangles within represent grid cells g1, g2, g3, and g4 partitioned from DS.

Definition 2 (Adjacent Grid Cells): Two grid cells g1 and g2 are called adjacent grid cells if their spatial regions are adjacent in the i-th dimension and identical in the other d-1 dimensions.

Definition 3 (Interior Point): Data points within a grid cell region are called interior points.

Definition 4 (Critical Point and Critical Region): Points within a grid cell region whose distance to any boundary of the grid cell is less than the density cutoff (see Definition 6) are called critical points, which also belong to interior points. The region covering critical points is called a critical region.

Definition 5 (Extension Point and Extension Region): Points outside a grid cell region whose distance to any boundary of the grid cell is less than the density cutoff are called extension points. The region covering extension points is called an extension region.

Definition 6 (Improved Local Density Calculation, Density Cutoff , and Density Cutoff Neighborhood): Based on the mathematical properties of the Gaussian function [11], when calculating the local density of a data object for a given cutoff distance dc, if a sample point' s distance to the specified data point exceeds dc, its influence on the local density calculation becomes negligible. This means each data point' s local density primarily depends on its nearby neighbors. Therefore, when computing local density, we can approximate it by considering only neighboring points within distance of the data point. The improved local density calculation formula is shown in Equation (3):

where dij represents the distance between data points i and j, and is called the density cutoff. The range within distance from a data object is called its density cutoff neighborhood.

After partitioning the data space, calculating a data object' s local density only requires considering objects within its own grid cell and adjacent grid cells, effectively avoiding the need to traverse the entire dataset and significantly reducing the computational workload.

Definition 7 (Grid Cell' s Extended -Space): Let g be a grid cell obtained from partitioning data space DS through spatial gridding. The extended -space of g, denoted as g+ , is the spatial region obtained by expanding each dimension' s boundary of g outward by distance .

As shown in [Figure 1: see original paper], the small rectangles bordered by dashed lines correspond to the extended -space of each grid cell. From Definitions 5 and 7, the extended -space of a grid cell equals the grid cell region plus its extension region.

Definition 8 (Data Partition): Let g+ denote the extended -space of grid cell g obtained from data space DS. Then, the data partition corresponding to grid cell g is defined as the set of all data objects in D that fall within g+ . Conversely, g is called the grid cell corresponding to data partition P.

From Definition 8, a data partition contains all data objects from the dataset distributed within the extended -space region of its corresponding grid cell. For a data object p in partition P, if p lies within the grid cell g corresponding to P, we can compute its local density estimate using the formula given in Definition 6.

---

3 SFSDP Algorithm Design and Implementation

The FSDP clustering algorithm must traverse all data objects when calculating local density and minimum distance, resulting in heavy computational load for single-node processing of large-scale data and low efficiency. The proposed SFSDP clustering algorithm addresses this by decomposing large-scale clustering tasks into smaller tasks that single nodes can process, significantly improving clustering speed. SFSDP execution consists of three phases:

Phase 1: Data Partitioning. Partition dataset D into several smaller data partitions with roughly uniform data volume based on its distribution in data space DS.

Phase 2: Local Clustering. Each computing node executes the improved FSDP clustering algorithm on its local partition data to obtain local clustering results.

Phase 3: Local Cluster Merging. Generate global clusters by merging and adjusting local clustering results.

3.1 Data Partitioning

To avoid traversing the entire dataset when calculating data objects'local density, Definition 6 introduces a new local density calculation method. The improved calculation only considers data objects within a radius centered on the data point, independent of other objects in the data space. Therefore, SFSDP can partition the data space into a spatial grid, allowing local density computation to consider only objects within the same and adjacent grid cells, greatly reducing time complexity.

For spatial grid partitioning, SFSDP employs a strategy based on the number of data objects within grid cells, using a KD-Tree [12] to partition data space into multiple grid cells with relatively balanced data object counts. This ensures load balancing among computing nodes and prevents data skew.

During dataset partitioning, critical points within grid cells have some neighboring data points outside their own grid cell within their density cutoff neighborhood. To compute these critical points' local density, some data objects must be assigned to multiple partitions simultaneously. As shown in [Figure 1: see original paper], if partition P corresponding to grid cell g1 only contains data objects within g1, critical point q' s local density calculation would be incorrect because its density cutoff neighborhood includes objects from adjacent grid cell g2. Therefore, partition P corresponding to g1 must also include some data objects from adjacent grid cell g2.

Data Partitioning Algorithm Implementation:

Input: D (dataset); max (maximum number of data objects in a grid cell)
Output: Partitions (data partitions after dataset division)

1. Partition data space DS into several equally sized, non-overlapping grid cells.
2. Calculate the number of data objects in each grid cell.
3. Initialize a queue Queue for spaces to be partitioned and add DS to the queue.
4. Initialize an empty grid cell collection G.
5. Pop the first spatial region S from Queue.
6. Calculate the number n of data objects in region S. If n is less than max, add S to collection G; otherwise, compute the variance of data objects in S across all d dimensions, select the dimension with maximum variance as the splitting dimension, divide S into two sub-regions S1 and S2 with balanced data volumes, and add S1 and S2 to Queue.
7. If Queue is empty, use G as the spatial grid partition for dataset D; otherwise, return to step 6.
8. Based on the partitioned spatial grid G, obtain data partitions of the dataset using Definition 8.

3.2 Local Clustering Within Partitions

After completing data partitioning, two modifications to the original FSDP algorithm are required to enable parallel local clustering on corresponding data partitions:

a) FSDP traverses the entire dataset when calculating local density and minimum distance. To enable independent computation of these values within partitions, we use Equations (3) and (4) to calculate data objects' local density and minimum distance within partition P.

b) FSDP requires manual selection of cluster centers. To eliminate human intervention, we use Equation (5) as an auxiliary function. The algorithm sets a cluster center threshold for local clustering and compares each data object' s value against this threshold, selecting objects with values exceeding the threshold as cluster center candidates.

Local Clustering Algorithm Implementation:

Input: P (data partition); dc (cutoff distance); threshold (cluster center threshold)
Output: Local clustering results for the data partition

1. For each point in P, compute its local density estimate using Equation (3).
2. For each point in P, compute its minimum distance and nearest neighbor using Equation (4).
3. For each point in P, compute its value using Equation (5).
4. For each point in P, if its value exceeds threshold, select it as a cluster center and assign it a cluster ID.
5. For each non-center point in P, assign it to the cluster of its nearest higher-density neighbor.
6. Compute the boundary density for each cluster.

3.3 Local Cluster Merging

To enable SFSDP to perform clustering independently on each data partition during the local clustering phase, the data partitioning stage creates overlapping partitions containing some common data objects. During the merging phase, the algorithm evaluates these common objects (critical and extension points) to identify local clusters requiring merging.

For convenience, assume C1 and C2 represent local clusters obtained from two overlapping partitions P1 and P2 (i.e., C1 P1, C2 P2, and P1 P2 Φ).

Theorem 1 (Merge Point Theorem): If there exists a data point p C1 C2 that is a core member in both local clusters C1 and C2, then C1 and C2 must be merged (points in C1 and C2 should belong to the same global cluster). Data point p is called a merge point of local clusters C1 and C2.

Proof: Core members correspond to cluster centers composed of high-density

data objects. The theorem follows directly from the definition of cluster core members.

Theorem 2: All merge points must be critical points or extension points within partitions. If data point p is a merge point of local clusters C1 and C2, then p must be a critical point or extension point within the partition.

Proof: Since p C1 C2 implies p P1 P2, point p lies in the overlapping region of P1 and P2. By the definitions of critical and extension points, Theorem 2 is proved.

Based on these theorems, determining merge points for local clustering only requires obtaining all critical and extension points within partitions and checking whether they satisfy Theorem 1. To acquire all critical and extension points, note that a partition' s critical points are also adjacent partitions' extension points, and only extension and critical points appear in multiple partitions. Therefore, we can group local clustering results by data objects, filter groups with more than one member as merge point candidates, and finally verify candidates against Theorem 1.

After identifying all merge points, we can determine the relationships between local clusters in different partitions based on which local clusters the merge points belong to. To conveniently assign global cluster IDs to local clusters, we use graph theory to represent local clusters as graph vertices and their relationships as edges, constructing a local cluster association graph. Local clusters belonging to the same global cluster form a connected subgraph. By generating a global cluster ID for each connected subgraph' s vertices, we complete the mapping from local to global cluster IDs.

Local Cluster Merging Algorithm Implementation:

Input: D (local clustering results in Key-Value format: (p, (pId, clusterId, flag))), where p is the data object, pId is the partition ID, clusterId is the local cluster ID, and flag indicates whether the object is a core or halo member.
Output: Global clustering results.

1. Group dataset D by data object p and filter objects appearing in multiple groups to obtain candidate merge point set CS.
2. Traverse CS and verify candidates against Theorem 1 using their flags (core member status) across partitions to obtain merge point set MS.
3. Build local cluster association set LS based on merge points' local cluster memberships, with elements in format (p1.clusterId, p2.clusterId) indicating p belongs to local cluster clusterId in partition p1 and also to local cluster clusterId in partition p2.
4. Initialize local cluster association graph G, add all local clusters as vertices, and generate self-pointing edges for each vertex.
5. Traverse LS and add edge (p1.clusterId, p2.clusterId) to G if not already present.

6. Generate a global cluster ID for each connected subgraph g in G and map all vertices in g to this global ID.
7. Update each data object' s cluster ID using the local-to-global mapping.

3.4 Algorithm Time and Space Complexity Analysis

1) Time Complexity Analysis

Assume the dataset contains n data objects. FSDP' s time consumption comprises two phases: (a) computing local density and (b) computing minimum distance. Both phases require two nested loops over the dataset, resulting in overall time complexity of $O(n^2)$.

SFSDP' s time consumption comprises three phases: (a) data partitioning, (b) local clustering within partitions, and (c) local cluster merging. Assuming M computing nodes process in parallel:

- **Data Partitioning:** Requires multiple passes over the dataset to determine partitions, with time complexity O(n).
- **Local Clustering:** If each grid cell contains at most m data objects, the dataset is divided into roughly n/m partitions, each containing approximately m objects (slightly more due to extension points, but this is negligible). Clustering on each partition costs $O(m^2)$, yielding overall complexity $O((n/m) \cdot m^2) = O(nm)$.
- **Local Cluster Merging:** Requires traversing the dataset to find merge points and update global cluster IDs, with complexity O(n).

Thus, SFSDP' s total time complexity is given by Equation (6). Considering additional communication costs between nodes, actual complexity will be slightly higher.

2) Space Complexity Analysis

FSDP requires $O(n^2)$ space to store pairwise distances between all data objects. In SFSDP, with at most m objects per grid cell, the dataset is divided into n/m partitions, requiring $O(m^2)$ storage per computing node. Therefore, SFSDP' s space complexity per node is $O(m^2)$, significantly reducing memory requirements compared to FSDP.

---

4 Experiments

4.1 Experimental Environment Setup

We built a Spark cluster using five ordinary machines, each with an Intel Core i5 2.7 GHz quad-core CPU, 8 GB RAM, and 512 GB hard disk. The software environment was CentOS 6.5, with all nodes interconnected via gigabit Ethernet switch. Spark version 2.3.0 was used.

4.2 Algorithm Accuracy Comparison

To verify SFSDP' s accuracy, we selected test datasets from and performed clustering using both SFSDP and FSDP. The accuracy comparison results are shown in .

As shown in , SFSDP' s clustering accuracy is slightly lower than the original FSDP. Analysis reveals two main reasons: (a) When calculating local density for extension points, some neighboring objects lie outside the same partition, causing underestimation and potential misclassification as halo members; (b) For some boundary points, their globally nearest higher-density neighbor may reside in adjacent partitions, leading to incorrect point assignment.

4.3 Algorithm Performance Comparison

To measure performance differences, we obtained taxi trajectory data from the Hong Kong University of Science and Technology Smart City Research Group (http://www.cse.ust.hk/scrg/), containing data from over 4,000 taxis. After processing, we generated coordinate files and randomly sampled five datasets: D1 (1,000 points), D2 (2,000 points), D3 (5,000 points), D4 (10,000 points), and D5 (20,000 points). Performance comparison results on these datasets are shown in [Figure 2: see original paper].

[Figure 2: see original paper] shows that FSDP requires less time for small datasets since it avoids partitioning and inter-node communication. However, as data scale grows rapidly, FSDP's runtime increases sharply, becoming ineffective at 20,000 points, demonstrating poor scalability. SFSDP, while incurring partitioning and communication overhead that makes it slower for small datasets, shows 平缓 time growth as data scale increases, with partitioning and communication costs becoming proportionally smaller. This demonstrates strong scalability.

4.4 Speedup and Scalability Analysis

To further validate scalability, we generated five test datasets from the taxi coordinate data: D1 (10,000 points), D2 (100,000 points), D3 (1 million points), D4 (2 million points), and D5 (5 million points). We measured speedup and scalability ratio across different node counts, with results shown in [Figure 3: see original paper] and [Figure 4: see original paper].

[Figure 3: see original paper] shows that SFSDP' s speedup increases with node count. For small datasets D1 and D2, the speedup curve' s slope decreases noticeably as nodes increase because computation time is small relative to communication overhead. For large datasets D3, D4, and D5, speedup increases nearly linearly.

[Figure 4: see original paper] shows that scalability ratio decreases as cluster size grows due to increased inter-node communication costs. However, larger

data scales yield higher scalability ratios, indicating SFSDP performs well on massive datasets.

5 Conclusion

To address FSDP' s low efficiency and poor scalability when clustering massive, high-dimensional data, this paper proposes SFSDP, a Spark-based parallel FSDP clustering algorithm. SFSDP first partitions massive datasets into balanced data partitions, then executes an improved FSDP algorithm in parallel across computing nodes, and finally merges local results into global clusters. Experimental results show that SFSDP maintains accuracy while demonstrating strong scalability and effective large-scale dataset clustering capability.

References

[1] Han Jiawei, Kamber M. *Data Mining: Concepts and Techniques* [M]. Translated by Fan Ming. Beijing: Mechanical Industry Press, 2001: 232-236.

[2] Zhou Z H. Three perspectives of data mining [J]. Artificial Intelligence, 2003, 143(1): 139-146.

[3] Omran M G H, Engelbrecht A P, Salman A. An overview of clustering methods [J]. Intelligent Data Analysis, 2007, 11(6): 583-605.

[4] Rodriguez A, Laio A. Clustering by fast search and find of density peaks [J]. Science, 2014, 344(6191): 1492-1496.

[5] Gan Wenyan, Liu Chong. An improved clustering algorithm that searches and finds density peaks [J]. CAAI Transactions on Intelligent Systems, 2017, 12(2): 229-236.

[6] Jiang Liqing, Zhang Mingxin, Zheng Jinlong, et al. Optimization of clustering by fast search and find of density peaks [J]. Application Research of Computers, 2016, 12(11): 3251-3254.

[7] He Ling, Wu Lingda, Cai Yichao. Survey of clustering algorithms in data mining [J]. Application Research of Computers, 2007, 24(1): 10-13.

[8] Bie R, Mehmood R, Ruan S, et al. Adaptive fuzzy clustering by fast search and find of density peaks [J]. Personal & Ubiquitous Computing, 2016, 20(5): 785-793.

[9] Zaharia M, Chowdhury M, Franklin M J, et al. Spark: cluster computing with working sets [C]// Proc of Usenix Conference on Hot Topics in Cloud Computing. [S.l.]: USENIX Association, 2010: 10-10.

[10] Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing [C]// Proc of Usenix Conference on Networked Systems Design and Implementation. [S.l.]: USENIX Association, 2012: 2-2.

[11] Barany I, Vu V H. Central limit theorems for Gaussian polytopes [J]. Annals of Probability, 2006, 35(4): 1593-1621.

[12] Bentley J L. Multidimensional binary search trees used for associative searching [J]. Communications of the ACM, 1975, 18(9): 509-517.

Note: Figure translations are in progress. See original paper for figures.