A Density-Grid Based Clustering Algorithm on Data Stream Using Resilient Distributed Datasets

Yuan Zhang(✉) and Jiongmin Zhang

Department of Computer Science and Technology,
East China Normal University, 200241 Shanghai, China
ecnu_zy@hotmail.com

Abstract. To find the clusters of arbitrary shapes rapidly from the sustainable growth of data stream, this paper proposes GDRDD-Stream algorithm. To capture the evolving characteristics of the data stream, this paper defines the effective time for the data points and design the eliminating strategy based on the effective time to remove the historical data. Secondly, we design the partitioning method based on resilient distributed datasets to balance the computing load between different nodes. Finally, we improve the traditional DBSCAN algorithm in order to compute in parallel between different partitions. The experimental results show that the proposed algorithm can cluster data stream distributed in arbitrary shape rapidly, capture the evolving behaviors of data stream, and its performance and quality are better than the CluStream algorithm.

Keywords: Data stream · Clustering · RDD · Spark · DBSCAN

1 Introduction

With the rapid development of Internet applications, an enormous amount of data stream arrived in real time, like stock data, network intrusion monitoring data and etc. Different with the static data, data stream [1] is massive, diverse, continuous, and rapid [2]. So the clustering algorithms on static data are no longer applicable to data stream. There have been some algorithms [3] which improved the traditional clustering algorithms in view of the characteristics of data stream: the clustering algorithms proposed in the articles [5, 6] are based on k-means which cluster data stream by single scanning. CluStream [7] is a framework, in which clustering process is divided into the online stage and the offline stage. And it could capture the evolution. But all of these algorithms are based on k-means, so which can only mine spherical clusters. D-Stream [8] applies the two-stage framework of the CluStream so as to capture the evolving process and is a density-based clustering algorithm to find the clusters of arbitrary shape. But its efficiency will decreased significantly for data stream of high dimension or rapid growth. So it is an urgent need for efficient clustering algorithm. Distributed cluster provides a revolution for large-scale data stream clustering. However, the clustering algorithms based on MapReduce [4] store the intermediate results on disks, and I/O operation is frequent with a certain delay. In order to improve the capability of

R. Khoury and C. Drummond (Eds.): Canadian AI 2016, LNAI 9673, pp. 316–322, 2016.
DOI: 10.1007/978-3-319-34111-8_38

parallel computing and the throughput of the cluster, Matei et al. proposed Resilient Distributed Datasets (RDD) [9]. RDD is a shared memory model without the need for frequent access to the disk so as to improve the computing performance of the cluster significantly.

This paper proposes a density-grid based clustering algorithm (GDRDD-Stream), which is based on the DBSCAN [10] algorithm on the Spark platform.

2 Design and Implementation of GDRDD-Stream

GDRDD-Stream algorithm achieves parallelization by storing data in resilient distributed datasets. Firstly, the whole data space is divided into the grids with the same size that is not less than 2*ε using the traditional spatial partitioning algorithm. Then these arrived data points are matched to the corresponding grids, and the elimination algorithm based on the effective time is implemented to filter out the grids and the data points. And then the RDD partitioning algorithm based on the number of the data points is implemented to partition. Finally, this algorithm improves the traditional DBSCAN algorithm to achieve parallelization, and adopts the basic idea of "parallel DBSCAN in partitions – DBSCAN in border points – merging results" to generate the clustering results. The flow chart of GDRDD-Stream algorithm is shown in Fig. 1.

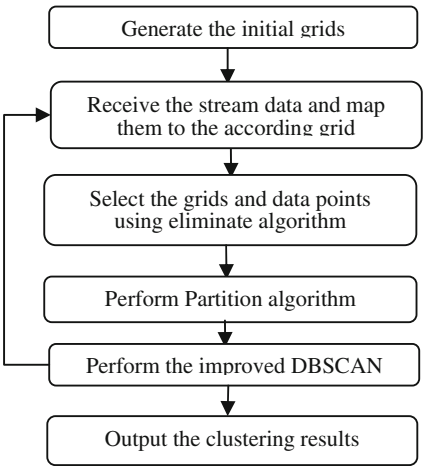

Fig. 1. The flow chart of GDRDD-Stream algorithm

2.1 Eliminate Algorithm Based on the Effective Time

The timeliness of the data affects the accuracy of clustering result. If clustering only the recent data stream, it cannot reflect the evolution of the data stream. Therefore, the concept of the effective time of data point is proposed in GDRDD-Stream algorithm. Data objects in the effective time will be clustered, weakening the impact of historical data and reflecting the evolution of data stream.

Definition 4 **Effective Time of the data point ($T_{effective}$):** Refers to a period of time that the data point impacts on the clustering results, that is, the longest time that the data points in memory can survive.

In order to further optimize the algorithm, GDRDD-Stream designs the elimination algorithm based on the effective time, which can reduce the amount of computation, and timely release of memory. The elimination rules are as follows:

- Record the update time of grid. If the update time of the grid over the effective time, delete its historical data and this grid does not participate in the cluster calculation.
- Detect every data point in the grid that if they exceed the valid time.

2.2 RDD Partitioning Algorithm

The grids selected by Elimination algorithm easily result in the uneven distribution of the data points in different partitions. GDRDD-Stream algorithm adopts the partitioning method based on the number of data points in grid. It balances the computing load between different nodes by merging the adjacent grids to ensure that the data points in each grid are relatively average.

- The minimum number that could be processed per partition can be calculated by the formula (1). The parameter processedPts is the points selected by Elimination algorithm. The parameter defaultParallelism, parallel degree of Spark, is set in Spark. Because the distribution of points in the partition is not even, use defaultParallelism*2 to make the number of the partition floating around the average value.

$$MinNum = \frac{count(processedPts)}{defaultParallelism \times 2} \tag{1}$$

- Merge the adjacent grids according to the minimum number of partition. Spark provides Partitioner interface, allowing developers to customize partitioning rule. Reallocate the grid IDs for the merged grids and generate the objects that inherits from the Partitioner interface. The RDD partition based on grid IDs is generated using the MapPartitionWithIndex interface in Spark.

2.3 The Improved DBSCAN Algorithm

GDRDD-Stream, based on DBSCAN, can discover clusters of arbitrary shapes and is of high clustering quality. Because the traditional DBSCAN algorithm needs to calculate the distances between all of data points, its time complexity is high. But the position of point in the data space is fixed so only the data points in the ε-neighbor need to be calculated. We could partition the data by the positions of them in dimension space. The data points in the same partition are of the close distance that most of the points in the same partition only find the ε-neighbor in this partition. And the border points(ε from the partition boundary) need to find the ε-neighbor from other partitions. Because the number of the border points is small and the clustering calculation in

partitions could be executed in parallel, the efficiency is significantly improved compared with the original algorithm. Based on the partitioning method, we improve the DBSCAN as following:

(1) DBSCAN algorithm is executed to cluster the data in every partition in parallel.
(2) Then, record the border points(ε from the partition boundary) and cluster the border points between different partitions.
(3) Finally, merge the clustering results.

It is important to note that the cluster identifier may be duplicate between different partitions. To avoid this situation, the identifier of the first point is allocated as the identifier of the new cluster because of the point identifier with the global uniqueness. For any two points in the border points, if they are not in the same grid and do not belong to the same cluster but their distance is less thanε, create a tuple (clusterId1, clusterId2) using their cluster identifiers and then merge them into the same cluster; if their distance is less thanεand one of them does not belong to any cluster, allocate this point with the cluster identifier of another point. And then merge all of the tuples and reassign the cluster identifiers for all data points.

If the total number of data points is n, the complexity of the traditional DBSCAN is $O(n^2)$. This algorithm improves the efficiency of clustering through parallel computing. If the partition number is m, the complexity is $O((n/m)^2)$, the efficiency may improve m^2 times nearly.

3 Experimental Results

We evaluate the evolution, quality and efficiency of GDRDD-Stream and compare it with CluStream. Our experiments are conducted on four PCs with 2.2 GHz CPU and 1 G memory. We evaluate GDRDD-Stream on CentOS 6.5 with Spark 1.3.1 and Scala 2.10.4 and use $T_{effective}$ = 40 s, eps = 30, Mpt = 8. We use two testing datasets. The first one is DS1 with 2-dimension which are synthetic. Its size is 80000 (the original distribution of the dataset is shown in Fig. 2). The second one is a real dataset used by KDD-CUP-99 [11] which contains 41 attributes and 5 categories.

3.1 Evolution Testing

We simulate the data stream based on DS1 at the speed of 1000 points per second to evaluate the evolution of GDRDD-Stream. We check the clustering results at four different times, including t1 = 40 s and t2 = 60 s. The clustering results are shown from Figs. 3 to 4. Since this algorithm is based on the effective time of data point and parallel in-memory computing, it can mine dynamically the evolution of the data stream and discover clusters of arbitrary shapes. The grids and data points are detected and eliminated periodically to release the memory resources in time.

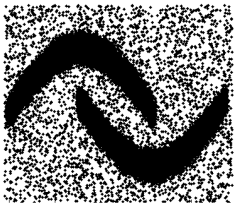

Fig. 2. Original distribution of the dataset DS1

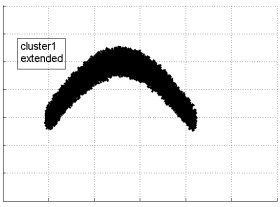

Fig. 3. Clustering result at t1 = 40 s

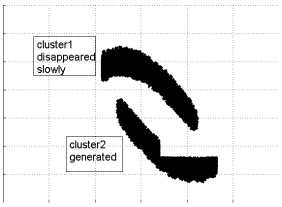

Fig. 4. Clustering result at t2 = 60 s

3.2 Correct Rate Comparison

The cluster purity is used to measure the quality of the clustering algorithm.

$$purity = \frac{1}{k}\sum_{i=1}^{k}\frac{|C_i^r|}{|C_i|} \tag{2}$$

In the formula (2), k is the number of clusters, $|C_i|$ is the number of the data points in the i^{th} cluster, $|C_i^r|$ is the number of the data points that are clustered rightly in the i^{th} cluster. The cluster purity of GDRDD-Stream and CluStream are compared on KDD CUP-99, shown in Fig. 5. The cluster purity of GDRDD-Stream is apparently higher than CluStream's at the time of 40 s, 60 s, and 80 s. It is caused by that CluStream can only discover the spherical clusters and need to know the value of k in advance while GDRDD-Stream can discover non-convex clusters. The clusters of arbitrary shape are generated and the number of clusters is unknown with flowing of data stream. GDRDD-Stream can allocate the network intrusion detection type to the right cluster while CluStream will allocate the different intrusion types into the same cluster. So the cluster purity of GDRDD-Stream is higher than CluStream.

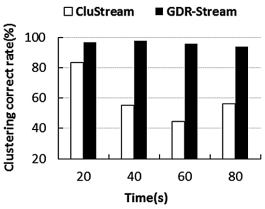

Fig. 5. Correct rate comparison on KDD CUP-99 (1000points/s)

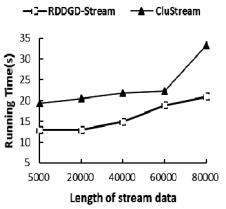

Fig. 6. Running time with changes of length of data stream

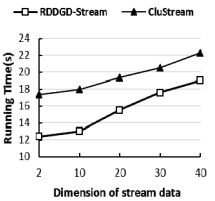

Fig. 7. Running time with changes of dimension of data stream

3.3 Efficiency Comparison

The efficiency comparison of GDRDD-Stream and CluStream with the growth of data stream length on KDD CUP-99 is shown in Fig. 6. The processing time of CluStream increases apparently with the extension of data stream length while the processing time of GDRDD-Stream increases slowly. The efficiency comparison of GDRDD-Stream and CluStream is shown in Fig. 7 with dimension from 2 to 40. The efficiency of GDRDD-Stream is higher than CluStream's. It is caused by that CluStream, based on complex distance calculation, will increase the calculation load with the growth of length and dimension of data stream. While GDRDD-Stream improves the efficiency through parallel computing between the RDD partitions based on grids.

4 Conclusion

In this paper, the GDRDD-Stream algorithm is proposed that are based on DBSCAN algorithm and resilient distributed datasets on Spark in view of the drawbacks of the current clustering algorithms of data stream. Experimental results show that GDRDD-Stream could capture the evolution of data stream and discover the clusters of arbitrary shapes in real time. And it has high efficiency without affecting the clustering results because of parallel computing using RDD. The algorithm makes high speed data stream clustering feasible without degrading the clustering quality.

References

1. Amineh, A., Teh, Y.W., Mahmoud, R., et al.: A study of density-grid based clustering algorithms on data streams. In: Proceeding of Eighth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), pp. 1652–1656 (2011)
2. Jonathan, A.S., Elaine, R.F., Rodrigo, C.: Data stream clustering: a survey. ACM Comput. Surv. **46**(1), 13:1–13:31 (2013)
3. Shifei, D., Fulin, W., Jun, Q., Hongjie, J., Fengxiang, J.: Research on data stream clustering algorithms. Artif. Intell. Rev. **43**, 593–600 (2015)
4. Jeffrey, D., Sanjay, G.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
5. Guha, S., Mishra, N., Motwani, R.: Clustering data streams. In: Proceeding(s) of 41st Annual Symposium on Foundations of Computer Science, pp. 359–366 (2000)
6. Ocallaghan, L., Meyerson, A., Motwani, R., Mishar, N., Gha, S.: Streaming data algorithms for high-quality clustering. In: Proceeding(s) of 18th International Conference, Data Engineering, pp. 685–704 (2002)
7. Aggarwal, C., Han, J., Wang, J., Yu, P.: A framework for clustering evolving data streams. In: Proceedings of the 29th International Conference on Very Large Data Bases, pp. 81–92 (2003)
8. Chen, Y., Tu, L.: Density-based clustering for real-time data stream. In: Proceeding of the ACM KDD 2007 Conference, pp. 133–142 (2002)
9. Matei, Z., Mosharaf, C., Tathagata, D., et al.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, p. 2 (2012)

10. Martin, E., et al.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of 2nd International Conference Knowledge Discovery and Data Mining (KDD 1996), pp. 226–231 (1996)
11. KDD dataset. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html