LITERATURE REVIEW: Parallel DBSCAN Clustering Algorithm using Apache Spark

Anousheh Shahmirza
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
Anoushehshahmirza@cmail.carleton.ca

October 2, 2019

1 Introduction

Analyzing big data is a very challenging problem today. Parallel computing is a type of computing architecture in which several processors execute or process an application or computation simultaneously. Parallel computing helps in performing large computations by dividing the workload between more than one processor, all of which work through the computation at the same time. By distributing the computations across hundreds or thousands of machines, the execution time reduces to a reasonable amount of time.

MapReduce framework has been devised to deal with big data in parallel. Google's MapReduce or its open-source equivalent Hadoop is a powerful tool for building such applications. With MapReduce, rather than sending data to where the application or logic resides, the logic is executed on the server where the data already resides, to expedite processing. This algorithm uses two user-defined functions which are called map and reduce functions [6]. Both map and reduce functions take a key-value pair as input and may output key-value pairs. This algorithm starts with applying a map operation to each logical record in the input to compute a set of intermediate <key, value> pairs, and then applying a reduce operation to all the values that shared the same key, to combine the derived data appropriately [2].

Apache Spark is an open-sourced programming model that supports a much wider class of applications than MapReduce. Apache Spark has a great performance for multi-pass applications that require low-latency data sharing across multiple parallel operations. This study is about applying the DBSCAN algorithm using the framework Spark. DBSCAN (Density-based spatial clustering of applications with noise) is an unsupervised learning data clustering approach that is commonly used in data mining and machine learning. Based on a set of points, DBSCAN groups together points that are close to each other based on a distance measurement and a minimum number of points. Also, this algorithm simply finds outliers point which are in low-density regions. This algorithm is popular since it can divide data into clusters with arbitrary shapes. Moreover, DBSCAN does not require the number of the clusters a priori as well as it is insensitive to the order of the points in the dataset [3]. However, applying DBSCAN with real-world data is challenging due to the size of datasets has been growing exponentially. This algorithm goes through each point of the database multiple times. The time complexity of the DBSCAN is O(n) which can be reduced to O(n log n) in some cases (n is the number of objects to be clustered). So the execution time for this algorithm highly increases when it comes to the massive dataset.

2 Literature Review

Presenting a parallel DBSCAN algorithm using the new big data framework Spark is receiving attention in recent years. As opposed to MapReduce based approaches for DBSCAN parallelization [5], [6], [1], there are few studies on DBSCAN clustering using Spark.

A pioneer algorithm [4] for presenting scalable DBSCAN algorithm with Spark, applies kd-tree to avoid communication between executors to reduce complexity from O(n2) to O(nlogn). The algorithm first reads data from the Hadoop Distributed File System (HDFS) and forms Resilient Distributed Datasets (RDDs), transforming them into data points. Certainly, this process is done in Spark driver. It then pushes all the data into multiple executors. Within each executor, partial clusters are built and sent to the driver. Each executor just performs its computation without communicating with others. Consequently, shuffle operations are prevented which costs a lot. Additional points are placed in each partial cluster. After all the partial clusters are collected through the shared variable accumulator, the algorithm identifies the clusters that are supposed to be merged by SEEDs. Merging is done in driver code too.

References

[1] Bi-Ru Dai and I-Chang Lin. Efficient map/reduce-based dbscan algorithm with optimized data partition. In *2012 IEEE Fifth International Conference on Cloud Computing*, pages 59–66. IEEE, 2012.

[2] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[3] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[4] Dianwei Han, Ankit Agrawal, Wei-Keng Liao, and Alok Choudhary. A novel scalable dbscan algorithm with spark. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1393–1402. IEEE, 2016.

[5] Yaobin He, Haoyu Tan, Wuman Luo, Huajian Mao, Di Ma, Shengzhong Feng, and Jianping Fan. Mr-dbscan: an efficient parallel density-based clustering algorithm using mapreduce. In *2011 IEEE 17th International Conference on Parallel and Distributed Systems*, pages 473–480. IEEE, 2011.

[6] Kyuseok Shim. Mapreduce algorithms for big data analysis. *Proceedings of the VLDB Endowment*, 5(12):2016–2017, 2012.