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 3, 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 [10]. 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 [7], [10], [4], [9], [1], there are few studies on DBSCAN clustering using Spark [5], [8], [6].

2.1 A novel scalable DBSCAN algorithm with Spark

A pioneer algorithm for presenting a scalable DBSCAN algorithm with Spark, first reads data from the Hadoop Distributed File System (HDFS) and forms Resilient Distributed Datasets (RDDs), transforming them into data points [5]. Certainly, this process is done in the Spark driver. It then pushes all the data into multiple executors. Within each executor, partial clusters are built and sent to the driver. There are no points that are shared between different partial clusters. The algorithm applies kd-tree to find the neighbours of a node. This is resulted to avoid communication between executors to reduce complexity from O(n2) to O(nlogn). Each executor only computes the points that belong to it. Otherwise, there would be a lot of overlap of computation between different executors. Consequently, shuffle operations are prevented which costs a lot.

This algorithm introduces the term: SEEDs, which are points that do not belong to the current partition. These are additional points that 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. These SEEDs serve as something like markers so that we can easily identify outer master partial clusters by using them and merge them into a bigger cluster. The SEEDs are not related to the locations. If the current point's index is beyond the range of current partition it is taken as a SEED. So the main goal on the executor side is to place SEEDs, and on the driver side, we dig out SEEDs and identify master partial clusters and merge them.

Taking advantage of Java Programming language, two data structures Hashtable and Queue are used in this algorithm. The complexity order of Put function in Hashtable is O(1 + n/K) where K is the hash table size. If K is large enough, the result is effectively O(1). Moreover, Method containsKey(key) is O(1). The algorithm executions generate the same result as the serial execuation [5].

2.2 A Parallel DBSCAN Algorithm Based On Spark

S_DBSCAN algorithm is divided into the following steps:

1) partitioning the raw data based on a random sample

2) computing local DBSCAN algorithms in parallel

3) merging the data partitions based on the centroid

As a result of the map task, partial clusters are generated. Merging stage follows four steps:

1) calculating the distance between every two partial clusters in the same partition; then use quicksort or heap-sort to find the minimum distance d_{min}

2) sort every min d_{min} to find the minimum value D_{min}

3) setting the threshold σ to merge partial clusters, and $\sigma << D_{min}$

4) creating a centroid_distance matrix and traversing every element in the matrix. If the distance is less than σ, then add them to the merge queue until every element is visited

S_DBSCAN Algorithm generates almost but exactly the same result as sequential DBSCAN [8].

2.3 Parallel DBSCAN Algorithm Using a Data Partitioning Strategy with Spark Implementation

This algorithm proposed a merging technique that maps the relationship between the local points and their bordering neighbours. The merging approach used in this study is very effective in reducing the time taken for the merge phase and very scalable with increasing the number of processing cores and the generated partial clusters [6].

The process starts with reading the original input data and partition data into multiple smaller and balanced sub-domains. The number of sub-domains is equal to the number of cores. The Spark driver creates a task-set request and the Task-Scheduler launches the tasks to executors. The Spark executors read data accordingly from disk and create partial clusters using kd-tree. The partial clusters are sent back to the driver at the end of the closure. Reading data from disk in executors instead of in the driver can break scalability barriers and achieve better performance. Each executor just performs its computation without communicating with other executors. While a partial cluster is created, the mapping relationship between a data point that is added in this partial cluster and the bordering neighbour is recorded by a Hashmap. The mapping relationship between points and their bordering neighbours is applied to merge partial clusters without communicating with other executors, which is very desirable in a parallelism environment. Moreover, search operation in the Hashmap data structure takes O(1) time if there is no collision. the data structure that holds this relationship is also effective in terms of storage space and this structure is designed as one part of the cluster structure itself.

After all the partial clusters are collected through a shared variable accumulator. The algorithm identifies the clusters that are supposed to be merged by the Hashmap. When one partial cluster Cn is generated by the help of some bordering points and the bordering points themselves are in another partial cluster Cm, these two clusters are going to be merged. The second case where we need to merge two partial clusters happens when two partial clusters share bordering points. We should merge them because they are supposed to be in one cluster if one sequential DBSCAN algorithm is run [6].

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] Yan Xiang Fu, Wei Zhong Zhao, and Hui Fang Ma. Research on parallel dbscan algorithm design based on mapreduce. In *Advanced Materials Research*, volume 301, pages 1133–1138. Trans Tech Publ, 2011.

[5] 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.

[6] Dianwei Han, Ankit Agrawal, Wei-keng Liao, and Alok Choudhary. Parallel dbscan algorithm using a data partitioning strategy with spark implementation. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 305–312. IEEE, 2018.

[7] 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.

[8] Guangchun Luo, Xiaoyu Luo, Thomas Fairley Gooch, Ling Tian, and Ke Qin. A parallel dbscan algorithm based on spark. In *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom)(BDCloud-SocialCom-SustainCom)*, pages 548–553. IEEE, 2016.

[9] Maitry Noticewala and Dinesh Vaghela. Mr-idbscan: Efficient parallel incremental dbscan algorithm using mapreduce. *International Journal of Computer Applications*, 93(4), 2014.

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