ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Special Issue 10, May 2016

Efficient Clustering on Big Data Map Reduce Using DBScan

Sailaja [1], Mushaq Ahmed D.M.[2]

P.G. Student, Department of Computer Engineering, AMC Engineering College, Bangalore, Karnataka, India[1]

Professor, Department of Computer Engineering, AMC Engineering College, Bangalore, Karnataka, India[2]

ABSTRACT: DBSCAN is a surely understood thickness based information clus-tering calculation that is generally utilized because of its capacity to discover self-assertively formed groups in loud information. Nonetheless, DBSCAN is difficult to scale which restrains its utility when working with expansive information sets. Strong Distributed Datasets (RDDs), then again, are a quick information preparing reflection made unequivocally for in-memory calculation of expansive information sets. This paper shows another calculation in view of DBSCAN utilizing the Resilient Distributed Datasets approach: RDD-DBSCAN. RDD-DBSCAN defeats the adaptability confinements of the customary DBSCAN calculation by working in a completely circulated style. The paper additionally assesses an execution of RDD-DBSCAN utilizing Apache Spark, the authority RDD usage.

KEYWORDS: DBSCAN; Apache Spark; data clustering; parallel systems; data partition; Resilient Distributed Datasets; MapReduce

I. INTRODUCTION

We live in a world that is turning out to be increasingly associated. A large number of little gadgets are conveyed all through the globe gathering data on climate conditions, the signs created by various machines and on human action when all is said in done. The data gathered by these frameworks can be utilized as a part of a wide range of courses, from foreseeing conceivable disappointments to prescribing the best answers for various issues in light of past encounters. A famous way to deal with conquer this trouble is machine learning, in particular, grouping calculations.

Among grouping calculations, Density-based Spatial Clustering of Applications with Noise (DBSCAN) is a standout amongst the most generally utilized. DBSCAN was initially created to work with regards to databases running in a solitary machine [2]. Since machine learning, including bunching, is particularly valuable in extensive information sets, it does not shock anyone that there has been a ton of examination into empowering DBSCAN to escape out of its single machine birthplaces. There is one specific range of examination which is of extraordinary enthusiasm for the connection of handling a lot of information however, the exploration on scaling DBSCAN through MapReduce.

MapReduce was presented in 2004 in an original paper distributed by J. Dignitary and S. Ghemawat [6]. The paper introduction duced a common nothing engineering that permitted the parallel preparing of a lot of information. The MapReduce worldview presented by the paper is currently generally utilized as a part of the business and is at the heart of the information preparing pipelines of a hefty portion of the organizations behind the beforehand specified PDAs and associated gadgets [7].

Since MapReduce is so generally utilized, an adjustment of DBSCAN for MapReduce turns out to be uncommonly helpful. From the numerous papers on this subject, B. R. Dai, and I. C. Lin in [8] and Y. He, et al. in [9] have both proposed varieties of DBSCAN that permit the calculation to keep running on top of the Apache Hadoop system, the most mainstream usage of the MapReduce worldview.

One of the enormous disadvantages of Hadoop's usage of MapReduce, is that the main correspondence that can happen between information handling ventures, in an information preparing pipeline, is through the record framework. This correspondence limitation presents a high cost, in the structure on dormancy, on top of the officially exorbitant calculation. M. Zaharia, et al. noticed this drawback and provided a solution in the form of an abstraction for in-memory computing: Resilient Distributed Datasets (RDDs).

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Special Issue 10, May 2016

Literature Survey

B. R. Dai, and I. C. Linin [8] and Y. He, et al. in [9] have both proposed variations of DBSCAN that allow the algorithm to run on top of the Apache Hadoop framework, the most popular implementation of the MapReduce paradigm.

One of the big drawbacks of Hadoop's implementation of MapReduce, is that the only communication that can happen between data-processing steps, in a data-processing pipeline is through the file system. This communication restriction introduces a very high cost, in the form on latency, on top of the already costly computation. This cost is acceptable for pipelines that only do transformations on the data. However,most machine learning algorithms (including DBSCAN) perform many iterations over the same set of data, with each step also depending on results on the previous steps.

This characteristic of machine learning algorithms has made those algorithms poor performers on the Hadoop platform [10]. M.Zaharia, et al. noticed this drawback and provided a solution inthe form of an abstraction for in-memory computing: Resilient Distributed Datasets (RDDs).

M. Zaharai et al. noticed that, while MapReduce effectively provides an abstraction on top of the computing resources of a cluster [10], iterative algorithms, in order to achieve reasonable levels of performance, need to also manage another of the cluster's resources: memory. Utilizing memory effectively allows iterative algorithms to avoid the performance penalties of reading data from the data store for each of their step, since the data is already present in memory.

M. Zaharai, et al. proposed Resilient Distributed Datasets(RDDs) as a solution to the shortcomings of MapReduce. RDDs, as defined by its creators, are an abstraction that "enables efficient data reuse in a broad range of applications"[10]. RDDs allow for algorithms to do coarse operations over data (e.g. map, filter, reduce) while also allowing for algorithms to manage the persistence of data in memory.

II. RELATED WORK

A. *Distributed Computing*

Customarily there have been two unique methodologies for preparing a lot of information. The primary methodology, when tasked with constantly expanding measures of information, builds the handling force of the specific machine with the undertaking of preparing information. This methodology is ordinarily alluded to as scaling vertically. The second approach, then again, rather than expanding the force of a solitary machine, builds the quantity of machines that are tasked with the preparing of the information.

In the most recent couple of years, scaling on a level plane has picked up noticeable quality as the favored way to deal with preparing a lot of information. The fundamental driver behind this inclination is the regularly diminishing expenses of PCs, alongside their perpetually expanding registering power. The decline of expenses and the expansion of force have made it conceivable to acquire the same measure of registering force from a few shoddy PCs cooperating, than from a solitary capable, however costly, machine. Consequently, most organizations that are occupied with preparing information have moved to an on a level plane scaling arrangement.

These complexities bring us to MapReduce. MapReduce provides a set of operations that allow the user to perform large-scale computations, without worrying about the complexities of distributing the computation throughout the cluster, or worrying about how to recover the computation in the case of failure.

B. *Resilient Distributed Datasets*

One of the drawbacks of the MapReduce paradigm is that it does not provide an efficient way to implement algorithms that have to perform multiple passes over the same data. In the map phase, each data unit is transformed and assigned a key (the key may be the same for multiple data units). In the shuffle phase, the data is partitioned according to its key and shipped to the reducers. Finally, in the reduce phase, the reducers generate some output for each group of data that shares the same key.

M. Zaharai et al. seen that, while MapReduce viably gives a deliberation on top of the registering assets of a bunch [10], iterative calculations, with a specific end goal to accomplish sensible levels of execution, need to likewise oversee one more of the group's assets: memory. Using memory adequately permits iterative calculations to keep away from the

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Special Issue 10, May 2016

execution punishments of perusing information from the information store for each of their progressions, since the information is as of now present in memory.
M. Zaharai, et al. proposed Resilient Distributed Datasets (RDDs) as an answer for the deficiencies of MapReduce. RDDs, as characterized by its makers, are a reflection that "empowers effective information reuse in a wide scope of uses" [10]. The primary commitment of RDDs, over different ways to deal with memory administration in a bunch, is that the arrangement of operations uncovered by RDDs takes into account RDDs to give effective adaptation to non-critical failure. While other group memory administration approaches use information replication (which is costly), RDDs monitor information ancestry. The benefit of RDDs' approach is that if data is lost for any reason, the lineage of the data can be tracked, and the lost data can be recomputed.

C. *The DBSCAN Algorithm*
DBSCAN is a density-based clustering algorithm. Densitybased clustering algorithms define a cluster as an area that has a higher data density than its surrounding area. In DBSCAN density is measured by analyzing whether a point has at least a minimum number of points (MinP ts) inside a given radius (Č). That is, if the Č neighborhood (points within Č) of a given point has exceeded a particular density threshold (MinP ts) that given point, and its neighbors, form a cluster. The DBSCAN algorithm is described in pseudo code in Algorithm 1.

Algorithm 1 The DBSCAN algorithm

Input: A set of points X = {p1, p2, . . . , pn}, the distance threshold Č, and the minimum number of points required for cluster MinPts.
Output: A set of labeled points X = {p1, p2, . . . , pn}, where each point has a flag corresponding to one of CORE, BORDER or NOISE and in the case of the flag being CORE or BORDER a corresponding cluster identifier.

```
1: clusterIdentifier ← next available cluster identifier
2: foreach unvisited point p ∈ X do
3: mark p' as visited
4: N ← GETNEIGHBORS(p, Č)
5:    if |N| < MinPts then
6:    p.flag ← NOISE
7:    else
8:       p.clusterIdentifier ← clusterIdentifier
9:       p.flag← CORE
10 :       foreach p'∈ N do
11:          if p' is not visited then
12:             mark p' as visited
13:             N' ← GETNEIGHBORS(p', Č)
14:             if | N'| ≥ MinPts then
15:                p'.flag ← CORE
16:                N ← N ∪ N '
17:             else
18:                p'.flag ← BORDER
19:            end if
20:         end if
21:         if p' does not belong to any cluster then
22:            p'.clusterIdentifier ← clusterIdentifier
23:            p'.flag ← BORDER
24:         end if
25:         clusterIdentifier← next cluster identifier
26:      end for
27:    end if
28: end for
```

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Special Issue 10, May 2016

III. DBSCAN USING RESILIENT DISTRIBUTED DATASETS

A. Overview

In this paper we introduce a new algorithm named RDD-DBSCAN. Our algorithm builds on the algorithm presented in [9], which parallelized DBSCAN using MapReduce. Our algorithm follows the same general outline from that paper, but introduces modifications that allow our algorithm to take advantage of the RDDs abstraction's ability to manage memory, improving the algorithm's efficiency. RDD-DBSCAN is described in Algorithm 2.

Algorithm 2 The RDD-DBSCAN algorithm

Input: A set of points X = {p1, p2, . . . , pn}, the distance threshold Č, the minimum number of points required for a cluster MinP ts and the maximum number of points per worker MaxPoints.

Output: A set of labeled points X = {p1, p2, . . . , pn}, where each point has a flag corresponding to one of CORE, BORDER or NOISE and in the case of the flag being CORE or BORDER a corresponding cluster identifier.

1: labeledP oints← Null
2:BR ← findMinimumBoundingRectangle(X)
3: P ← EvenlyP artition(BR, 2Č, MaxP oints)
4: **foreach** partition ∈ P **do**
5: partition ← partitionexpandedBy(Č)
6: points ← pn ∈ partition
7: labeledP oints ∪DBSCAN(points, Č, MinP ts)
8: **end for**
9: aliases ← IdentifyAliases(P, labeledP oints, Č)
10:clusters ← all unique clusters in labeledP oints
1 1:RelabelP oints(aliases, clusters, labeledP oints)

After the partitions are found, RDD-DBSCAN enters the local clustering stage. In the local clustering stage, RDD-DBSCAN performs local clustering for each partition using the traditional DBSCAN algorithm. In order to efficiently perform the local clustering, RDD-DBSCAN will load all the data points for a given partition into memory. Only then, will the local clustering be done. Once the local clustering completes, RDD-DBSCAN makes use of the RDDs abstraction's data management operations to persist the results of the clustering into memory. Once all global clusters have been identified in this way, all the points are relabeled with the correct global cluster and the algorithm concludes. Fig. 1 gives an overview of the steps necessary to perform RDD-DBSCAN.

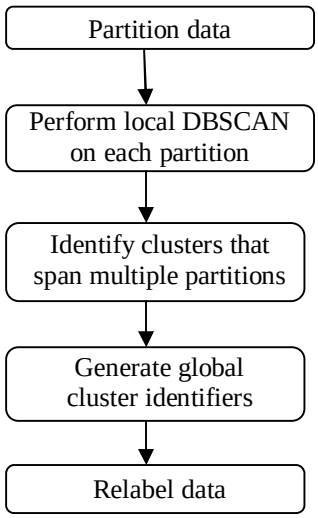

Fig. 1. RDD-DBSCAN overview

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Special Issue 10, May 2016

One important restriction of RDD-DBSCAN is that it assumes that the data points to be clustered can be represented in two dimensions. The reason for this restriction is that RDD-DBSCAN uses a two-dimension representation of the space which contains the data to come up with an efficient partitioning scheme. If the data cannot be represented in a two dimension space, then partitioning fails. There is a wide body of literature dealing with dimensionality reduction, so it is expected that this restriction does not limit the applicability of RDD-DBSCAN.

IV. EVALUATION

A. Platform

To evaluate the performance and correctness of RDD-DBSCAN we chose to implement RDD-DBSCAN using Apache Spark, the popular implementation of the RDDs abstraction. Since its introduction, Apache Spark has become extremely popular and has had tremendous growth: as of 2014, Apache Spark is the most active open source project in the Big Data ecosystem [16]. As such, Apache Spark is the obvious target for the implementation of RDD-DBSCAN.

B. Language

Apache Spark is implemented in Scala but it also has bindings to Java and Python, allowing for algorithms to be implemented in any of those languages. We chose to implement RDD-DBSCAN using Scala, because Apache Spark itself is written in this language; using Scala provided the best interoperability with the platform.

C. Test Data

A data set is an integral part of the evaluation of clustering algorithms, and for the evaluation of RDD-DBSCAN we chose to use a synthetic data set. A synthetic data set allowed us to better observe the behavior of RDD-DBSCAN under different conditions that are harder to control in non-synthetic data. The data set consisted of one million entries, organized into five different clusters with five different centers.

D. Correctness

Since DBSCAN is mostly deterministic, it is possible to verify that a distributed version of DBSCAN is correct by comparing the results of both the original version and the distributed version. If the results of both are identical, then the distributed version is correct. To verify the correctness of RDD-DBSCAN, we generated a smaller version of the test data with the tools mentioned in the previous paragraph. The test data was then clustered using *scikit-learn*'s implementation of DBSCAN and with RDD-DBSCAN. The output of both executions of the algorithm was identical, so we were able to verify that RDDDBSCAN was correct.

E. Complexity

Given that DBSCAN, as described in Algorithm 1, has a complexity of O(n2) we chose not to use that algorithm in our local DBSCAN phase. Instead we used an R-tree as a helping data structure to perform the GETNEIGHBORS query in Line 13 of the algorithm. With the help of R-tree the complexity of local DBSCAN comes down to O(n log n). One important observation is that the R-tree used in our implementation is completely held in memory, with each R-tree containing all the points for any given partition. Holding the R-tree in memory allows for very fast execution of the algorithm, but it constrains the number of points that may be included in a partition, to those that can be held in memory. In our experiments, exceeding the memory capacity of our nodes resulted in the Java Virtual Machine entering long garbage collection cycles and eventually crashing with out of memory errors. Apache Spark exposes the amount of memory available for the execution of a given algorithm, so it is possible to automatically detect this condition before it happens.

F. Hardware

Our implementation of RDD-DBSCAN was evaluated using a cluster of five virtual machines running inside the Amazon Web Services environment. Each machine was an instance of an *m3.large* model, with each instance having 2 vCPUs of type Intel Xeon E5-2670 v2, 7.5 GiB of memory and 32 GB of SSD Storage.

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Special Issue 10, May 2016

V. EXPERIMENTAL RESULTS

The important goal of our experiment, was to determine whether RDD-DBSCAN would scale linearly as the amount of available computing power was increased. Fig.2 confirmed that, as expected, the time taken by RDDDBSCAN decreased as the number of datasets available for computation increased. One important drawback of distributed algorithms is that, as their parallelism increases, so does the cost of communication between the computing nodes.
Our experiment show that first, the time taken by computing tasks completely dominates the costs of communication between nodes; and second, that as the number of nodes increases, the percentage of time taken by the computing tasks remains constant. These results indicate that RDD-DBSCAN is not significantly affected by communication costs, which is not surprising given that Apache Spark only exposes a very limited set of intra-node communication mechanisms. Larger partitions, in turn, tend to contain a larger number of points which have to be loaded into memory by the computing node when they are processed.

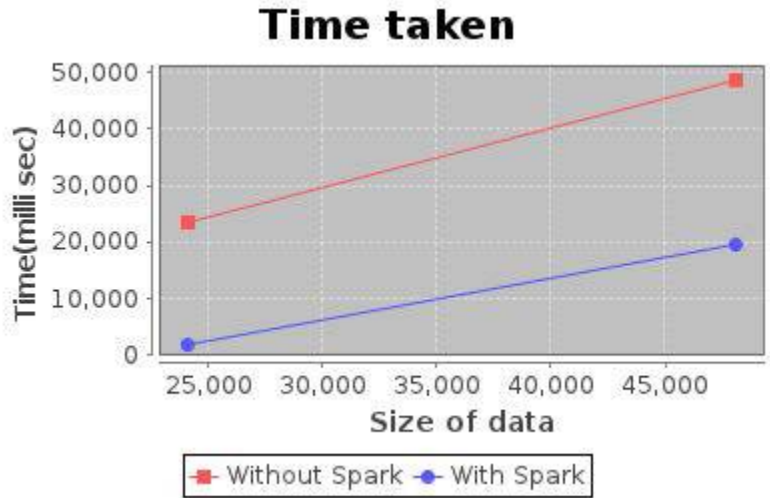

Fig 2: Graph of comparison on the basis of size of data

VI. CONCLUSION

This paper presented RDD-DBSCAN, a conveyed version of the DBSCAN calculation utilizing Resilient Distributed Datasets that creates an indistinguishable result to that of the first DBSCAN. We portrayed the contemplations that go into the configuration of the calculation, and in addition the benefits of this methodology over other comparable methodologies. We gave a point by point clarification of every progression of the calculation furthermore portrayed how these strides can be deciphered into a real usage. At that point, we demonstrated tentatively that RDD-DBSCAN scales as the quantity of hubs in a given group scale while producing the same results as the successive variant of DBSCAN.

REFERENCES

[1] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise.” in KDD, vol. 96, no. 34, 1996, pp. 226–231.
[2] M. Chen, X. Gao, and H. Li, “Parallel dbscan with priority r-tree,” nin Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on. IEEE, 2010, pp. 508–511.
[3] M. M. A. Patwary, D. Palsetia, A. Agrawal, W.-k. Liao, F. Manne, and A. Choudhary, “A new scalable parallel dbscan algorithm using the disjoint-set data structure,” in High Performance Computing, Net-working, Storage and Analysis (SC), 2012 International Conference for. IEEE, 2012, pp. 1–11.
[4] Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008
[5] B.-R. Dai and I.-C. Lin, “Efficient map/reduce-based dbscan algorithm with optimized data partition,” in Cloud Computing (CLOUD), 2012