International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 04 Issue: 07 | July -2017 www.irjet.net p-ISSN: 2395-0072

Parallel *k*NN for Big Data using Adaptive Indexing

Tejal Katore[1], Prof. Dr. Suhasini Itkar[2]

[1]Post Graduate Scholar, Dept. of Computer Engineering, P.E.S Modern College of Engineering, Pune, India
[2]Professor, Dept. of Computer Engineering, P.E.S Modern College of Engineering, Pune, India

---

Abstract - *k Nearest Neighbor is frequently used in classification methods. kNN algorithm defines the class membership of the given element. kNN when used in context with large data, does not perform well. So multiple techniques were introduced to execute kNN parallely and enhance its performance. Along with this MapReduce programming model was used which was suitable for distributed approaches. The different reference algorithms were given as follows HzknnJ, HBNJ, RankReduce which compute kNN on MapReduce. Data preprocessing, Data partitioning and computation are the three common steps for kNN computation. For all given solutions only the partitioning technique differs. Adaptive Indexing is a indexing paradigm where index creation and reorganization takes place automatically and incrementally. It was used along with the RankReduce algorithm which helps knn to exec more efficiently.*

***Key Words*: Hadoop Block Nested Loop kNN (H-BNLJ), Hadoop z value (H-zkNNJ), k Nearest Neighbor, MapReduce, Performance Evaluation, RankReduce.**

1. INTRODUCTION

k Nearest Neighbor is widely used as a classification or clustering method in machine learning or data mining[1]. The k-Nearest Neighbor algorithm (k-NN) [2] is considered one of the ten most significantly data mining algorithms . It is an lazy learner which do not need absolute training phase. The method requires that all of the data instances are stored and unseen cases classified by finding the class labels of the k closest instances to them[3]. To determine how close two instances are, several distances can be computed. This operation as to be performed for all the input examples against the whole training dataset.

Given R is a point and S is set of reference points, a k nearest neighbor join is an operation which for each point in R, discovers the k nearest neighbor in S. The data points are divided into training set and testing set, also called unlabeled data. The aim is to find the class label for the new points. For each unlabeled data, a kNN query on the training set will be performed to estimate its class membership. This process can be considered as a kNN join of the testing set with the training set. The basic idea to compute a kNN join is to perform a pairwise computation of distance for each element in R and each element in S . The difficulties mainly lie in the following two aspect: (1) Data Volume (2)Data Dimensionality. A lot of work has been dedicated to reduce the in-memory com-putational complexity [1]. These works mainly focus on two points: (1) Use indexes to decrease the number of distances need to be calculated. These indexes can hardly be scaled on high dimension data. (2) Use projections to reduce the dimensionality of data. But the maintenance of the accuracy becomes another issue. Despite these efforts, there are still significant limitations to process kNN on a centralized machine when the amount of data increases [4],[10],[11].

Only distributed and parallel solutions are proved to be powerful, for large dataset . MapReduce is a flexible and scalable parallel and distributed programming paradigm which is specially designed for data-intensive processing. MapReduce is a parallel programming model that aims at efficiently processing large-datasets. It consists of:(1) representing a key-value pair (2)defining map function

(3)defining reduce function. Here we introduce the reference algorithms that compute kNN over MapReduce. These algorithms are based on different methods, but follow a common work-flow which consists three ordered steps:(1)data pre-processing (2)data partitioning (3) kNN computation.

2. LITERATURE REVIEW

kNN is based on a distance function that measures the difference or similarity between two instances. kNN using centralized approach was not able to perform for large inputs. So a new approach to execute it parallelly was developed. There are various existing solutions to perform the kNN operation in the context of MapReduce are given.

The approach HBNLJ[1] consists of two phases. The data set is divided into a certain blocks of particular size. The data is partitioned such a that an element in a partition of R will have its nearest neighbor in only one partitioned of S. Two partitioning strategies that enable to separate the datasets into independent partitions, while preserving locality information, are proposed. H-zkNNJ [1],[4], which use size based partitioning strategies, have a very good load balance, with a very small deviation of the completion time of each task. In H-zkNNJ, the z -value transformation leads to information loss. The recall of this algorithm is influenced by the nature, the dimension and the size of the input data. More specifically, this algorithm becomes biased if the distance between initial data is very scattered, and the more input and M , the number of hash functions in each family. Since they are dependent on the dataset, experiments are

needed to precisely tune them. In, the authors suggests this can be achieved with a sample dataset and a theoretical model. The first important metric to consider is the number of candidates available in each bucket. Indeed, with some poorly chosen parameter values, it is possible to have less than k elements in each bucket, making it impossible to have enough elements at the end of the computation. HzknnJ uses Locality Sensitive Hashing[7][8][9].

Rank Reduce [1],[5], with the addition of a third job, can have the best performance of all, provided that it is started with the optimal parameters. The most important ones are W ,the size of each bucket, L , the number of hash families. Increasing the number of families L greatly improves both the precision and recall. However, increasing M , the number of hash functions, decreases the number of collisions, reducing execution time but also the recall and precision. Overall, finding the optimal parameters for the Locality Sensitive Hashing part is complex and has to be done for every dataset.

Special type of distance [8], [4] is adaptive indexing. It is specifically addressed kNN queries in high-dimensional space and has since proven to be one of the most efficient and state-of-the-art high dimensional indexing techniques available for exact kNN search. In recent years, iDistance has been used in a number of applications. In a set of one-dimensional distance values, each related to one or more data points, for each partition that are all indexed together in a single standard B + -tree. The algorithm was motivated by the ability to use arbitrary reference points to determine the similarity and dissimilarity between any two data points in a metric space, allowing single dimensional ranking and indexing of data points no matter what the dimensionality of the original space [8].

3. SYSTEM ARCHITECTURE

Processing Steps The following scheme consists of three basic steps:

1) Pre-processing
   i. Remove column names
   ii. Move to HDFS
   iii. Feature Extraction
   iv. Clean Data
   v. Divide into training and testing set
2) Partitioning
3) kNN Computation

In iDistance algorithm indexing was added with Rank Reduce in between. From this the reference was taken and the implemented system also had indexing with Rank Reduce but in shuffled order. Firstly the indexing is performed and then the Rank Reduce is executed.

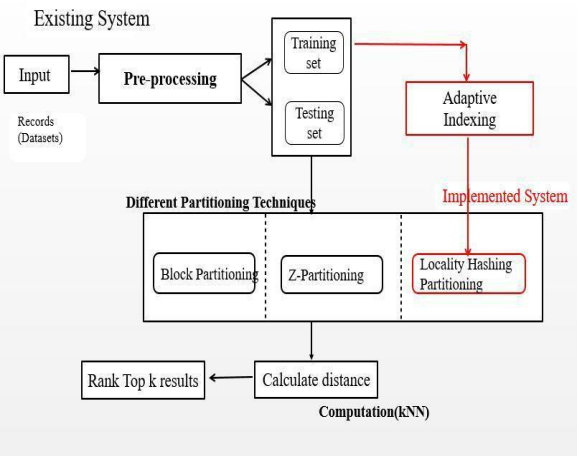

Fig -1: Architecture Diagram

1.Pre-processing- The data is transformed from its original form to the data that is beneficiary. Only the required data is kept and remaining is removed. It further consists of few steps. A).Remove the column names- The attributes or column names of the data set are removed. B).Move to HDFS- The data set is moved over the Hadoop Distributed File System. C) Feature Extraction- Extracting the selected features from the given data . D) Clean Data-After selecting the features the remaining data is discarded.
E) Divide into training and testing set- The data set is divide into training as well as testing data set. Training data set is the labelled data which consists of class membership. Testing data set is the unlabeled data which is to be processed.

2.Partitioning- While processing data on MapReduce, we need to divide the data set into independent pieces, called as partitions. Partitioning is the process of dividing the data into blocks, regions, buckets, etc. All the algorithms use different partitioning strategies. Partition is done using 2 different strategies: (1) Distance based Partitioning Strategy

(2) Size based Partitioning Strategy. In distance based partition the space is divided into disjoint cells while in size based partition the space is divided into equal size partition. The algorithms are divided under both strategies.

(3). kNN Computation - The reducers perform the computation. The mappers divide the data set into numbers of blocks and the output of these is given to the reducers. Then the reducers sort the points according to the distances.

A DataSets

The dataset used is named as "Airline On-Time Statistics and Delay Causes". The dataset consists of records of airlines which include all details related to flights. It is packaged in yearly chunks from 1987 to 2008. It consists of 29 columns from which 19 columns are selected. The

column name "cancelled tickets" is used as labeled column. The size of the data set is 12 GB. It consists of 52 billion records[12].

B. Hardware

1) Memory: 8GB
2) Processor: Intel (R) Pentium (R) CPU B950 @2.10 GHz
3) Hard disk: 64 GB

C. Software

1) Cloudera :Hadoop framework
2) Operating System: CentOs
3) Eclipse 4.2.2 and above
4) vi editor

D. Performance Parameters

The performance of the system is measured using different parameters among which time is the important factor. The number of mappers and reducers for each algorithm are kept same. The value of k is randomly taken. The time required for each algorithm to execute is in seconds.

E. Results

Precision

Algorithms	k=10	k=15	k=20
Block	0.57	0.56	0.55
Z-value	0.7	0.31	0.2
LSH	0.85	0.73	0.68
Adaptive Indexing	0.9	0.77	0.7

Table 1: Precision value for each algorithm

The comparison between all four algorithms is done. Some advantages and shortcomings of all algorithms are observed. HBkNNJ is trivial to implement. It breaks easily but is good for tiny dataset. H-BNLJ is easy to implement but has a Very large communication overhead. It performs well for small and middle datasets. While H-zkNNJ is fast and more precise. But it requires large disk space and gets slower for high dimensional dataset. RankReduce performs best among all algorithms. It is fast and can be used for high dimensional data. Adaptive indexing yeilds best results among all algorithms in terms of precision. For different values of k precision values are taken.

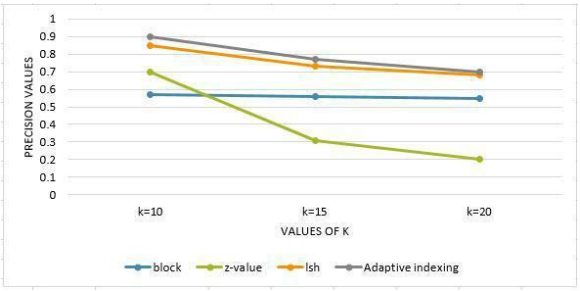

Fig. 2. Performance of algorithm in seconds

Similarly time of execution is reduced in Adaptive Indexing. Compared to all remaining algorithms Adaptive Indexing takes less time for all values of k.

Time			
Algorithms	k=10	k=15	k=20
Block	28.3	25.06	26.32
Z-value	30	30.1	32
LSH	23	25.3	24
Adaptive Indexing	17.2	21	19.8

Table 2:Execution Time for each algorithm in minutes

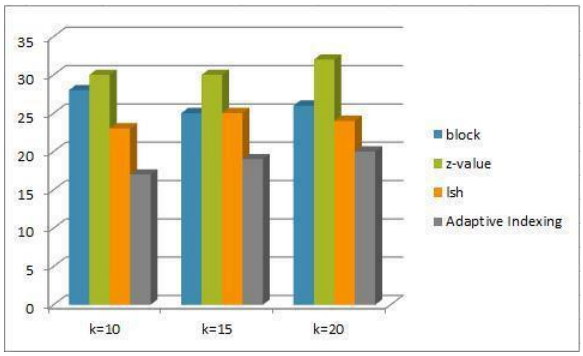

Fig. 3. Time required for execution in minutes

4. CONCLUSION

In the given paper we implemented the existing solution for kNN operation in context of Map Reduce[6]. All solutions follow three main steps that are pre-processing, partitioning, and computation. Different reference algorithms are implemented. Depending upon the time and complexity required Adaptive Indexing is found to be efficient. The parallel implementation has helped to improve the efficiency of the kNN. Using indexing over hashing may reduce the time

required for iterations. And it has contributed in performance of the algorithm.

5. ACKNOWLEDGEMENT

Working on the topic "Parallel kNN for Big Data using Adaptive Indexing" was a source of immense knowledge to me. I would like to express my sincere gratitude towards Prof. Suhasini Itkar for her guidance and valuable support thought out of research work. I acknowledge with a deep sense of gratitude , the encouragement and inspiration received from our staff members and friends. Last but not the least I would like to thank my parents for their love and support.

6. REFERENCES

[1] Ge Song, Justine Rochas , Lea El Beze and Fabrice Huet , K Nearest Neighbor Joins for Big Data on MapReduce: a Theoretical and Experimental Analysis, in Proceedings of IEEE Transactions on Knowledge and Data Engineering 1041-4347 2016.

[2] T. M. Cover and P. E. Hart, Nearest neighbor pattern classification, IEEE Transactions on Information Theory, vol. 13, no. 1, pp. 2127.

[3] X. Wu and V. Kumar, Eds.," The Top Ten Algorithms in Data Mining",Chapman Hall/CRC Data Mining and Knowledge Discovery, 2009.

[4] G. Song, J. Rochas, F. Huet, and F. Magouls, Solutions for Processing K Nearest Neighbor Joins for Massive Data on MapReduce, in 23rd Euromicro International Conference on Parallel, Distributed and Network-based Processing, Turku, Finland, Mar. 2015.

[5] W. Lu, Y. Shen, S. Chen, and B. C. Ooi, Efficient processing of k nearest neighbor joins using map reduce, Proc. VLDB Endow., 2012.

[6] A. Stupar, S. Michel, and R. Schenkel, RankReduce - processing k- nearest neighbor queries on top of map reduce, in In LSDS-IR, 2010.

[7] C. Zhang, F. Li, and J. Jestes, Efficient parallel knn joins for large data in mapreduce, in Extending Database Technology, 2012.

[8] C. Yu, R. Zhang, Y. Huang, and H. Xiong, High-dimensional knn joins with incremental updates, GeoInformatica, 2010.

[9] B. Yao, F. Li, and P. Kumar, K nearest neighbor queries and knn-joins in large relational databases (almost) for free, in Data Engineering (ICDE), 2010 IEEE 26th International Conference on, March 2010, pp. 415.

[10] http://stat-computing.org/dataexpo/2009/the-data.html.