Metric KNN Joins Using Spark

Xiaohai Cheng

***Abstract*— k nearest neighbor join(kNN Join) refers to finding the k nearest neighbor vectors in another data set R for each object in the data set S by using the nearest neighbor formula and the distance calculation formula between vectors in two data sets.kNN Join has a wide range of applications, so it has received attention.However, as a combination of k nearest neighbor join queries and join calculations, processing kNN joins of high dimensional data is quite time consuming.To handle larger data sets, the Hadoop and Spark frameworks have been the tool of choice for parallel and distributed computing in recent years.In this paper, we present the vertical decomposition data for processing kNN connections using Spark.We propose to first approximate the vector piecewise aggregatio and the symbol aggregation approximation SAX, then decompose the vector vertically to satisfy the distributed operation, calculate the distance between different vectors in each partition, and then merge the partitions to meet the conditions. Calculate the vector and finally find the true distance between the vectors.The specific process is described in the implementation section of the text. The experimental results show that the proposed method improves the computational speed under the premise of ensuring the accuracy of the experimental results.**

***Index Terms*—kNN Join, Hadoop, Spark, symbolic aggregation, vertical partition**

I. INTRODUCTION

kNN Join or its variant operations have been broader application prospects and application value, including friend recommendation[1], pattern recognition[2], clustering[3], image similarity matching[4], outlier detection[5], spatial database[6] and other related fields. kNN Join is an asymmetric join operation that returns each of the points in the relationship R with the k nearest points in the other relationship S. However, most traditional algorithms use spatial indexes such as B+ trees[7], R-trees[8] or z-order curves[9] to improve the performance of kNN connections, but for high-dimensional data sets of large amounts of data, these methods can be very time consuming.

For such data-intensive similarity calculations, the MapReduce framework[10] has become the primary choice for big data processing. Recently, some scholars have proposed parallel kNN connection algorithms using MapReduce, such as H-BNLJ, H-BRJ[11] and PGBJ[12]. Since PGBJ can filter non-similar kNN data in advance, PGBJ technology is better than H-BNLJ and H-BRJ in performance. However, as the data set dimension increases and the amount of data increases, its computational efficiency is greatly reduced. Nowadays, the similarity calculation of high-dimensional vectors also has the SAX method[13], which is very suitable for processing large-scale data sets.

Manuscript received Oct 08, 2018

Xiaohai Cheng, School of Computer Science & Software Engineering, Tianjin Polytechnic University, Tianjin, 300387, China

Spark[14] is also a big data processing framework developed in recent years. It is a reference for MapReduce distributed big data processing, and it is optimized by the memory framework to make it have stronger data processing capabilities. In this paper, we propose an efficient parallel algorithm based on the Spark platform to improve the parallel computing efficiency of kNN Join and reduce the running time. We first need to normalize each dimension vector in the original R and S data sets to the [0,1] interval, and then perform horizontal dimensionality reduction through the PAA representation and SAX, and then partition by the vertical decomposition technique. The intersection calculation or distance calculation is performed between the vectors. The specific calculation method is the histogram intersection and the Euclidean distance. Finally, the approximate pairs satisfying the conditions in the distance are filtered out, the distance between the vectors is recalculated, and k objects of each vector of the S data set are selected, wherein less than all of the k vectors are returned.

II. PROBLEM FORMULATION AND RELATED KNOWLEDGE

In this section, we introduce the definition of the basic variables and formulas in this article. We focus on the core concepts of this paper -- symbolic aggregation approximation and vertical decomposition methods.

2.1 Problem formulation

Histogram intersections and Euclidean distances are often used in image retrieval as similarity measures. When the histogram intersection is used as a measure of image similarity, the overlap between the two histograms in each dimension is added, and if their histogram intersection is large, the two images are considered to be similar. The calculation of the Euclidean distance is calculated by calculating the distance between two vectors, and if their distance in the feature space is small, the images are considered to be similar. The kNN join result set of this paper is calculated by these two methods.

Definition 1 (Histogram intersection) Let H be a set of normalized image histograms (N-dimensional vector h, $(\forall h \in H : 0 \le h_i \le 1)$).Given two normalized histograms h and q, we define the histogram intersection as a measure of the similarity between them:

$$|h,q| = \sum_{i=1}^{N} \min(h_i, q_i) \qquad (1)$$

Using histogram intersections assumes that different dimensions are irrelevant. This metric has been shown to be superior to the Euclidean distance, mainly because it reduces the computation of extraneous vectors in the query results.If the histograms are very similar, the intersection of the two histograms is about N, because

$\forall i,1 \le h_i \le N : \min(h_i, q_i) \cong h_i$ and T(h)=N.If the histograms are significantly different, their intersection will be small.

Definition 2 (Euclidean distance)Let V be a set of N-dimensional feature vectors v $(\forall v \in V : 0 \le v_i \le 1)$,The Euclidean distance between two vectors v and q of dimension N is defined as follows:

$$D_E(v,q) = \sqrt{\sum_{i=1}^{N}(v_i - q_i)^2} \qquad (2)$$

Definition 3 (k nearest neighbor) Given an object r, a data set S and an integer k, find the k nearest neighbor points of S from K as KNN(r,S), and also find a set of k objects from S as $\forall o \in KNN(r,S)$, $\forall s \in S - KNN(r,S), |o,r| \le |s,r|$ 。

Definition 4 (kNN Join) Given two data sets R, S and an integer k, the kNN join of R and S is expressed as $R \rhd S$, Each of these objects $r \in R$, From each nearest neighbor in S is represented as $R \rhd S = \{(r,s) | \forall r \in R, \forall s \in KNN(r,S)\}$ 。

2.2 Symbol aggregation approximation

2.2.1 PAA representation of high dimensional vectors

PAA[15]is a dimension reduction technique and is widely used in time series processing and trajectory related research. It is to divide the original high-dimensional data into equal dimensions, and calculate the approximate distance of the original vector by using the distance formula given by the following definition 5. The vector used in this paper is a sequence-independent vector, and the order of the dimensions does not affect the calculation of the Euclidean distance. When necessary, you can rearrange the dimensional order of the vectors and then use a piecewise aggregation approximation to represent the high-dimensional vectors.

Definition 4 (PAA representation) Given an n-dimensional vector R, divide its dimensions into equal parts, and let N be the dimension after the division. $\overline{R} = \bar{r}_1, \bar{r}_2, ..., \bar{r}_N$ is a representation of N dimensions, which has a relationship of $\overline{R}_n = \bar{r}_1 \cup \bar{r}_2 \cup ... \cup \bar{r}_N$ and $\bar{r}_i \cup \bar{r}_j = \phi$ .Then the vector R has a PAA expressed as $R_n = (\bar{r}_1, \bar{r}_2, ..., \bar{r}_N)$, Each dimension vector is represented as shown below:

$$\bar{r}_i = \frac{N}{n} \sum_{j=\frac{n}{N}(i-1)+1}^{\frac{n}{N}i} r_j \qquad (3)$$

2.2.2 SAX representation of high dimensional vectors

Definition 5(Symbol aggregation approximation) Given the N-dimensional vector R after the PAA representation, divide each dimension into an approximate interval of a symbol representation. If A1, A2, A3, ..., An are used to approximate the estimate, it can be expressed as $R_{SAX} = \{A_1, A_2, ..., A_N\}$.

Definition 6(Degree of polymerization λ) Assuming that the dimension of R is n and the dimension after its PAA representation is N, the degree of aggregation λ is defined as $\lambda = n/N$ 。

Given two vectors R and S, the Euclidean distance after their PAA representation can be defined as:

$$D_{PE}(R,S) = \sqrt{\lambda \sum_{i=1}^{N}(R_{Ni} - S_{Ni})^2} \qquad (4)$$

The histogram intersection can be defined as:

$$D_{PH}(R,S) = \lambda \sum_{i=1}^{N} \min(R_{Ni}, S_{Ni}) \qquad (5)$$

It has been shown that the distance represented D_{PE} by PAA is the lower limit of the original Euclidean distance D_E .

$$D_{PE}(R,S) \le D_{PE}(R,S) \qquad (6)$$

Similarly, PAA is the upper limit of the histogram intersection. That is:

$$D_{PE}(R,S) \ge D_{PH}(R,S) \qquad (7)$$

Given two vectors R and S and their SAX representation R_s and S_s , We can define new distances as follows:

$$SE(R,S) = \lambda \sqrt{\sum_{i=1}^{N} dist(\hat{R}_{Ni}, \hat{S}_{Ni})^2} \qquad (8)$$

The definition of the histogram intersection is as follows:

$$SH(R,S) = \lambda \sum_{i=1}^{N} \min(\hat{R}_{Ni}, \hat{S}_{Ni}) \qquad (9)$$

$dist(\hat{R}_{Ni}, \hat{S}_{Ni})$ is a subfunction that calculates the distance between any two pairs of symbols, where the distance is obtained by looking up the table. It is easy to prove that the distance between the SAX representations (SE) is the lower bound of the distance between the PAA representations and D_{PE} is the lower bound of the Euclidean distance; according to the transitivity, SE is the lower boundary approximation of the Euclidean distance:

$$SE(R_s^{\lambda}, S_s^{\lambda}) \le D_E(R,S) \qquad (10)$$

Similarly, the distance of the histogram intersection represented by SAX is the upper limit of the distance represented by PAA, and D_{PH} is the upper bound of the intersection of histograms. According to transitivity, SH is the upper bound of the intersection of histograms:

$$SH(R_s^{\lambda}, S_s^{\lambda}) \ge D_H(R,S) \qquad (11)$$

III. IMPLEMENTATION

In this part, we introduce a novel Spark-based kNN Join method. The main problem we have to solve is the reliability of the operation results, the running time and the utilization of computer resources when the amount of data is large. Since the self-connection and the RS connection are similar, the specific implementation of this paper only considers the case where the data set is self-joined.

3.1 SAX aggregation approximation in Spark

At this stage, we have found the maximum and minimum of all vectors. We need to normalize each dimension of the vector to the [0,1] interval by the normalization formula, and use the PAA as defined above according to the SAX degree of polymerization. Indicates that the formula is dimension-reduced, and then the vector represented by the PAA is SAX-represented. If you need to estimate the distance calculation in the partition, you need to calculate the sum of

the sum vector T(h), and finally return the key value pair <SAX,(T(h), pid)>, pid is the original vector. The number.
The reduce phase consists of a key-value pair returned from the map, which aggregates keys with the same key-value pair. The value in the key-value pair will contain multiple vectors. The original aggregate is Iterable, which we convert to a list.

3.2 Vertical decomposition and intra-partition calculation in Spark

In this section, the methods for vertically decomposing data sets and intra-partition calculations are introduced. This process is mainly for the vertical decomposition of the data after SAX polymerization in 3.1. For example, suppose the original vector is N-dimensional, SAX indicates that the back vector is r-dimension, and the number of partitions needs to be t, then each vector is equally divided into t parts, and the dimension in each partition is r/t. The vector is then vertically decomposed from low to high. Algorithm 1 gives a pseudo-code representation of the vertical decomposition. The 2-11th lines of the original SAX key-value pair decomposition process. The r/t dimension vectors in each partition are approximated by SAX distance, and the remaining N-r/t dimension vectors are estimated by estimation. The sixth line averages the estimated vectors, and the eighth line divides each SAX vector into vectors. The ninth line requires a vector representation format to return.

Alogrithm 1 RDD-SAVD-Partition(SAX,newValue,SAID)

SA,SAID：Symbolic aggregate approximation and its id；
newValue:Estimation of N-M dimensional vectors for each partition
begin
1. pivot_id = 0;
2. for i <— 0 to SAX.length **do**
3. **for** j <— 0 to newValue.length **do**
4. array_add <— getArrayAdd(newValue);
5. **end for**
6. array_a**ve** <— getArrayAve(array_id);
7. array_add = 0;
8. new_sax <— getNewSax(SAX);
9. result <— (pivot_id,(sax_id,new_sax,array_ave));
10. pivot_id += 1;
11. end for
12. emit(result);
end

The calculation in the partition needs to perform the reduce aggregation on the vertically decomposed vector, and aggregate the vectors with the same vd_id into one. Then calculate the distance between the two in each partition, and finally output the first k vectors of each vector. Algorithm 2 gives the pseudo code computed within the partition. Each partition is a list collection. The 1-8 lines calculates the distance between the vectors in the list through a double for loop. The third line calculates the histogram intersection distance of the two vectors, and the fourth behavior stores the temporary variables. The sixth line extracts the first k corresponding to the vector.

Alogrithm 2 RDD-SAVD-Aggregation(listValue)

listValue：Aggregation vector for each partition
begin
1. for i <— 0 until listValue.length **do**
2. **for** j <— 0 until listValue.length if i!=j **do**
3. distance <—
histogram_intersection_sax_distance(listValue(i),listValue(j));
4. comp += Tuple2((listValue(i)._1,listValue(j)._1),distance);
5. end for
6. result <— getK(comp);
7. comp.clear();
8. end for
9. emit(result);
end

3.3 Each partition takes the intersection and the final kNN Join operation

When each partition completes the kNN Join calculation, each vector of each partition will have a k-to-key-value approximation data pair. We need to perform intersection operations on these data pairs, filter out the vector pairs that satisfy the condition, and then recalculate the distance between the original vectors by matching the table with the original vector id in the sax vector pair in the previous cache(). Since the SAX aggregated vector filters out a portion of the data pair, the pair of data that needs to be calculated will become less. Algorithm 3 below shows the pseudo code for this process. The 1-9 lines calculates the distance between the original vectors. We need to find the original vector set corresponding to each sax in the symbol aggregation pair <saxidi, saxidj>, and calculate the original distance and the set internal vector between the two sets. The distance between the two. The 2-5 lines calculates a vector pair inside a saxid, and the 6-9 lines calculates the distance between the vectors between saxidi and saxidj.

Alogrithm 3
RDD-SAVD-ResultIntersection(sax_ids,sax_map,old_vector)

sax_ids：Take the SAX pair after the intersection,
sax_map: SAX with the original vector id
old_vector: Original vector
begin
1. for tmp_list <— sax_map(sax_ids._1) **do**
2. **for** tmp_list1 <— sax_map(sax_ids._1) if tmp_list._2 != tmp_list1._2 **do**
3. distance <—
histogram_intersection(old_vector(tmp_list._2),old_vector(tmp_list1._2))
;
4. result += Tuple2((tmp_list._2,tmp_list1._2),distance);
5. end for
6. **for** tmp_list2 <— sax_map(sax_ids._2) **do**
7. distance <—
histogram_intersection(old_vector(tmp_list._2),old_vector(tmp_list2._2))
;
8. result += Tuple2((tmp_list._2,tmp_list2._2),distance);
9. end for
10. end for
11. emit(result);
end

IV. EXPERIMENTAL CONFIGURATION AND DATA SETS

In order to verify the effectiveness of the proposed method in the Spark-based kNN Join method. We implement the algorithm of this paper separately with Spark architecture and Hadoop architecture, and compare it with the existing improved Hadoop framework based on the same experimental conditions. The experimental parameters were changed, the changes of their performance were observed, and the experimental results were analyzed to obtain objective conclusions. Our experiments were done in Hadoop 2.6.4 cluster and Spark 2.1.1 cluster. The cluster has 10 nodes. The configuration of each node is as follows: core: 4 cores, memory: 6 GB, disk: 500 GB, operating system: Linux CentOS release 6.2 (Final). In Spark, one of them is the Master node and the other 9 are the worker nodes.
The experimental data used primarily in this paper is from the data set provided in [15] and can be downloaded from http://corpus-texmex.irisa.fr/. We used some of the data and

modified it, with 128- and 960-dimensional datasets for downloading data, and 256 and 512-dimensional datasets generated by 960-dimensional datasets. Due to the needs of the experiment, we will number each line of the original data by linux command, which can make the calculation more efficient.

V. EXPERIMENTS

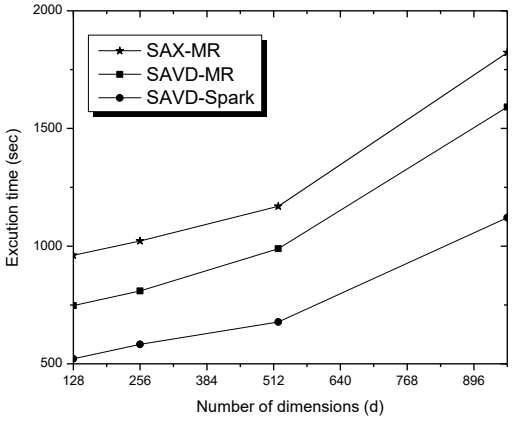
In the existing distributed high-dimensional similarity connection model based on MapReduce, we improved its algorithm into kNN Join algorithm and established a model to compare with it. We mainly adjust the size of the dimension (from 128 to 960) and adjust the size of k in the kNN Join in the experiment (from 10 to 50). These methods will be the key factors in the following experiments to evaluate the efficiency of the algorithm.

5.1 Experimental evaluation

In this section, we mainly compare the execution time of our proposed algorithm. A total of two sets of comparative experiments were set up, which are the effect of dimensional change on experimental performance and the effect of parameter k on experimental performance in kNN Join. The influence of dimensional change on experimental performance includes: 1. Comparison of original improvement method[13] and the method of this paper on MapReduce platform and Spark platform; 2. Comparison of Euclidean and histogram methods. In order to facilitate comparison, in addition to the experimental comparison of Euclidean and histogram dimensions, we all use the histogram to find the result of kNN Join. The experiments in this paper are all tested by self-joining of data sets.

5.2 The effect of dimensional change on experimental performance

In order to reflect the effect of our proposed method, we improved the original SAX-based high-dimensional data similarity connection method, so that it can perform kNN Join operation, and use it as a reference method to compare different dimensions with our proposed algorithm.In the experiment, other parameters were adjusted to a fixed value, the k value was 10, the SA polymerization degree SA_DP was 8, and the vertical decomposition polymerization degree DP was 16. Figure 1 is a graph showing the effect of changes in dimensions on the run time of the three methods.



.Figure 1:Effect of dimensionality .

Through experiments, it is found that with the increase of the dimension, the time used by the three methods is increasing, and the overall trend is consistent. The MapReduce-based method is more time-consuming than the Spark-based method. Of course, the original method takes a relatively long time. The proposed method has different performances on different platforms. However, the Spark-based SAVD algorithm is used in different dimensions and has the best experimental results.

Because our similarity method has two kinds of Euclidean distance and histogram intersection. We performed a comparison of different methods on the same platform, but the histogram intersection has fewer calculation steps and higher computational efficiency, so it takes less time. Since the histogram intersection in this paper proposes two methods, after many experiments, we find that the second method is better than the first one. So in the following we all use the most efficient histogram intersection for comparison. Figure 2 is a graph showing the effect of the change in dimension on the Euclidean and histogram intersection run time.

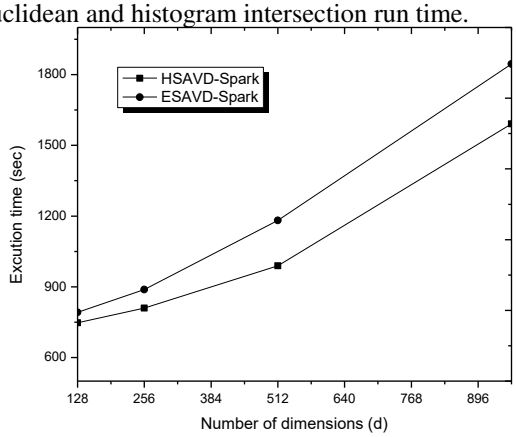


Figure 2:Effect of dimensionality

From the trend graph drawn from the experimental data, we can find that as the dimension increases, the time spent on kNN Join is gradually increasing, but the calculation of the Euclidean distance does not take more time for the method of calculating the histogram intersection.

CONCLUSION

The kNN Join of high-dimensional data is computationally intensive, especially when the amount of data is very large, we need to find a simple way to reduce the amount of computation. In this paper, we tested the method with different data sets. A lot of experimental research shows that our method is more efficient, and Spark-based kNN Join is the shortest time. Our method has a good effect on improving kNN Join of high dimensional vectors.

REFERENCES

[1] Xianke Zhou,Sai Wu,Gang Chen,Lidan Shou.KNN processing with co-space distance in SoLoMo systems.Expert Syst.Appl.2014,41(16):6967-6982.

[2] M. Muja, D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data", Pattern Analysis and Machine Intelligence IEEE Transactions, vol. 36, no. 11, pp. 2227-2240, 2014.

[3] C. Böhm, F. Krebs, "Supporting KDD applications by the k-nearest neighbor join", DEXA, 2003.

[4] Giuseppe Amato,Fabrizio Falchi,Local Feature based Image Similarity Functions for kNN Classification.ICAART (1) 2011:157-166

[5] S. Ramaswamy, R. Rastogi, K. Shim, Efficient algorithms for mining outliers from large data sets. SIGMOD Rec, vol. 29, no. 2, pp. 427-438, 2000.

[6] G. R. Hjaltason, H. Samet, "Distance browsing in spatial databases", ACM Transactions on Database Systems (TODS), vol. 24, no. 2, pp. 265-318, 1999.

[7] D. Comer, "Ubiquitous b-tree", ACM Comput. Surv, vol. 11, no. 2, pp. 121-137, 1979.

[8] A. Guttman, "R-trees: A dynamic index structure for spatial searching", SIGMOD Rec, vol. 14, no. 2, pp. 47-57, 1984.
[9] G. M. Morton, "A computer oriented geodetic data base; and a new technique in file sequencing", Technical Report 1966.
[10] Jeffrey Dean,Sanjay Ghemawat.Mapreduce: Simplified data processing on large clusters[J].Commun. ACM,2008,51(1):107-113.
[11] Chi Zhang,Feifei Li,Jeffrey Jestes.Efficient parallel knn joins for large data in mapreduce[C]//EDBT 2012:38-49
[12] Wei Lu,Yanyan Shen,Su Chen,et al.Efficient Processing of K nearest neighbor joins using MapReduce[J]. PVLDB,2012,5(10):1016-1027.
[13] Youzhong Ma,Xiaofeng Meng, Shaoya Wang.Parallel similarity joins on massive high-dimensional data using MapReduce[J].John Wiley and Sons Ltd, 2016,28(1):166-183.
[14] Apache, “Apache hadoop,” https://spark.apache.org
[15] Luo W, Tan H, Mao H, Ni LM. Efficient similarity joins on massive high-dimensional datasets using mapreduce[C]//IEEE Computer Society :Washington, DC,US,2012:1-10.