Distributed Stream Processing of RNN Query in Mobile Computing

Siqi Xu[1], Changqing Ji[1,*], Yanran Zhuang[1], Sunying Gao[1], Nianpeng Yang[1], Jingguo Yan[2], Xin Zhang[3]

[1] School of Physical Science and Technology, Dalian University, Dalian, 116622, China

[2] School of Information Engineering, Dalian University, Dalian, 116622, China

[3] School of Mechanical Engineering, Dalian University, Dalian, 116622, China

Email: jcqgood@gmail.com

Keywords: Reverse Nearest Neighbor, Big Data, MapReduce, Cloud Computing

Abstract. Reverse Nearest Neighbor (RNN) queries are a pipeline of complimentary problems, and have aroused a vast concern in the world in the past few years, such as location based services, profile based marketing, resource allocation and traffic monitoring system etc. Now the one of the most important disadvantages for the RNN is that it has inherent sequential nature and using for memory algorithm, which limits its use in data processing of large scale spatial data queries. Scalable algorithms for Reverse Nearest Neighbor queries in distributed environment are the key problem in this paper. First of all, we investigate the SRNN initialization query method based on the inverted grid index. Then, Eager-SRNN has effective treatment on the problem of the scalable Multi-dimensional RNN. Eager-SRNN tries to prune spatial objects step by step as soon as they are accessed. Beyond that, SRNN algorithm is the first attempt for the exact scalable RNN algorithms in a distributed environment on multi-dimensional datasets. An evaluation which we proved through a lot of experiments has been widely applied on the new method of the synthetic data scalability and the performance.

Introduction

In recent years, smart phones, laptop and tablets remarkably have started to carry sensors like GPS, RFID, Camera and Bluetooth, etc [1]. Because of the large scale spatial data are increasing rapidly, it makes people worried when they want to analyze the centralized algorithms or computations on large volumes of multi-dimensional spatial data. Therefore, the distributed algorithms for spatial data processing which will remove people worries, so there is an urgent need in the contemporary [2].

As a typical spatial query algorithm, RNN query retrieves the objects whose nearest neighbors include the query point q which is a typical spatial query algorithm. It has been welcomed by intelligent navigation, modern communications, traffic control, profile based marketing, resource allocation and other areas [2]. In order to make it easy to understand, we cite a simple example, if a restaurant wants to attract more crowds and obtain higher profit, it will change its service to cater to more customers [3].

This paper expands on the basis of an earlier published conference papers [2], [5] and [7] in several substantial aspects. At first, Eager-SRNN, a new optimization strategy is proposed. In Eager-SRNN, we attempt to prune spatial objects as soon as they are visited. Second, we propose an accurate and complete proof of correctness for the SRNN algorithms. Third, compared with the state-of-the-art algorithms, the SRNN is equally complex. At last, in order to evaluate the performance of our proposed algorithms objective and correctly, we added more experiments.

Related Work

After the development over the years, the way to solve the exact RNN query through technologies has evolved from doing a linear search of all spatial objects. Korn et al, firstly [3]

* Changqing Ji is corresponding author.

defined the RNN query. As a hotly debated issue attracted many researchers' attention. The R-tree index such as KM [3], it is not perfect, because it cannot guarantee the results no omissions. TPL relies mainly on recursively filtering the data by finding perpendicular bisectors between the query point and its nearest object [4]. Therefore, it needs to overcome a lot of difficulties to study to make the RNN algorithms scalable in a distributed environment.

Due to the central single node has no or very few computation and storage capability. One way to solve large scale RNN query is to use multiple machines simultaneously, and perform the spatial index and query in distributed [5]. At present, MapReduce[1] is caused by the enthusiasm of the people as a popular alternative for massive-scale parallel data analysis in shared-nothing clusters [6]. Each MapReduce job consists of two stages: Map function and Reduce function.

Guoren Wang et al. proposed the design and evaluation of ComMapReduce which improve the MapReduce framework with lightweight communication mechanisms, supporting applications for large scale datasets [8].

In this paper, our goal is to propose a scalable approach that enables efficient processing of large-scale, accurate and multi-dimensional spatial RNN named SRNN (Scalable Reverse Nearest Neighbor).

Handing Scalable RNN Query

In this Section 3.1, we give a definition of SRNN firstly. Then, we describe Eager-SRNN for continuous MapReduce jobs over data streams. A Scalable Reverse Nearest Neighbor query is formally defined as below:

Definition 1

Scalable Reverse Nearest Neighbor query (SRNN): Given a large set of spatial objects P in an n-dimensional large scale space D and a query object q, SRNN query is distributed to retrieve all the objects $p \in P$ which have q as their nearest neighbors (NN). Formally, $SRNN = \left\{ p \in P \middle| Dist(p,q) < Dist(p,q^{'}) \right\}$, where $Dist()$ is a distance metric and p' is arbitrary point in P in addition to p.

In Lazy-SRNN query process [5], the verification step cannot fetch the output of a filter step until it has finished executing and committed its final output to disk. So, we have to wait for a long time until all of the results are displayed, which not only wasting our time, but also we cannot find our mistakes through the processing. In this section, we present an improved algorithm for incremental and support online aggregation of scalable Reverse Nearest Neighbor queries (Eager-SRNN, for short). Although SRNN can stream quite directly using modified version of the MapReduce framework such as MapReduce Online [9] and Yahoo's Open Sourced S4 [10] etc. They can only be used for share-nothing system that is not what we wanted. We need a new streaming system to support global RNN pruning.

Firstly, in filter step, we recall the PCT algorithm to find a partial result of the nearest neighbors of a query point q in each map round by round. We also use pre-clustering based inverted grid index to eliminate nodes that may lead to RNN results. The radius starts from a small non-null value as the initial and shrinks when the completeness is improved through iterations (the number of objects we examine within the radius remains roughly the same). If candidate kNN is found when some rounds finished, we eliminate some data points that have no influence on the result through half-space pruning. The Eager-SRNN algorithm initializes a kNN list L_{cnd} by inserting the pruned kNN result objects. Then, we will immediately send the intermediate result set L_{cnd} to reducer and run next round continually with no pause. When reducer gets the L_{cnd}, we could call verification step to get the points p_i are the real RNN of q. Note that in Lazy-SRNN, and a reduce task cannot fetch the output of a filter task until the filter has finished executing and committed its final output to disk. So the results which are worked out by the SRNN presented to the user as soon as possible.

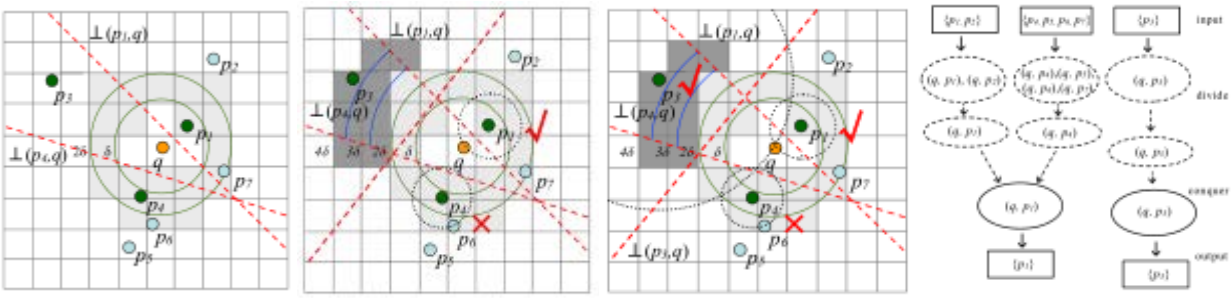
Algorithm 1 Eager-SRNN

Input: Inverted grid index G ,Query point q , $p_i \in S_{cn}$

Output: $SRNN(q)$ //the SRNN of query point q

1: **Filter():**
2: Initialize set $S_{cnd} = PCT(G, q, k)$
3: Initialize $TG = G$
4: **while** $TG > \infty$ **do**
5: **for** each point $p_i \in S_{cnd}$ **do**
6: $TG = Half - planesTrim(TG, q, p_i)$
7: **if** $p_i \in \perp(p_i, q)$ **then**
8: $S_{cnd} = S_{cnd} - \{p_i\}$//pruned point
9: **end if**
10: **end for**
11: **end while**
12: $L_{cnd} add(p_i)$
13: return L_{cnd}
14: Verification()
15: **Verification():**
16: **for** each point $p_i \in L_{cnd}$ **do**
17: **if** $q \in PCT(G, p_i, k)$ **then**
18: $L_{cnd} = L_{cnd} - \{p_i\}$ //false hit
19: **end if**
20: **end for**
21: return L_{cnd}

In addition to reduce many points by introducing the half-plane pruning just like Lazy-SRNN, our Eager-SRNN query method can also be parallel just like a stream. For example, as shown in Figure 1(a), in first round any point in $\perp(p_1, q)$ and $\perp(p_4, q)$ like p_2, p_7, p_5 and p_6 can be pruned by half-pruning. So we get the partial candidate results of NN of q are p_1, p_4 and will immediately send the middle result set $L_{cnd} = \{p_1, p_4\}$ to reducer. In Verification step, upon testing for their nearest objects, it turns out that only p_1 and p_3 are reverse nearest neighbors while p_4, has p_6 as its nearest object since distance$(p_6, p_4) <$ distance(q, p_4) the dotted circles in Figure 1(b)). Finally, Reduce outputs results set which is $\{(q, p_i)\}$ as early returns to users. Then, in mapper we run next round continually with no pause and only consider the points in the unpruned area and find the nearest neighbor of q in the next pruning. As shown in Figure 1(c), we find p_3 as a NN of q in the second round. We send it to reducer and call parallel verification to get the point p_3 are the real RNN of q as latterly returns to users. The termination of condition of pruning is that there are no points in unpruned area. As a result, $SRNN(q) = \{p_1, p_3\}$ undergone several different phases continually.



(a) Filter-1 (b) Verification-1&Filter-2 (c) Verification-2 (d) Divide-Conquer

Figure 1. Illustration of Eager-SRNN

Analytical Comparison of Eager-SRNN Algorithms

The following formulas define the equations of our cost-based optimization of SRNN. The parallel processing for SRNN is:

In the map phase, *m* mappers process S_m bytes of data. The majority of the extra time spent is related to reading the input data from disk. D_s is the average bytes/sec. That data of MapReduce task (mapper or reducer) can deal with local disks. S_m bytes of data will be read in parallel by mappers, which are able to read D_s bytes per second each.

Then in the shuffling phase, S_S bytes of data are shuffled to *r* reducers. The majority of the shuffling cost is related to shipping the data between distinct machines through the network. D_r is the ratio of data actually shipped between distinct machines relative to the total amount of data processed. Thus, the total amount of data to be shipped is $S_{cnd} \cdot D_r$ bytes. The data will be received in parallel by reducers, each one receiving in average N_s bytes per second.

In the reduce phase, S_{cnd} bytes of data are analyzed by *r* reducers. The reducers will read from disk s bytes in parallel at the individual cost of D_s bytes per second.

Thus, we can get the formula (1),

$$S(q) = CostM(m, S_m) + CostS(s, S_{cnd}) + CostR(r, S_{cnd}) = \frac{S_m}{m} \cdot \frac{1}{D_s} + \frac{S_{cnd} \cdot D_r}{r} \cdot \frac{1}{N_s} + \frac{S_{cnd}}{r} \cdot \frac{1}{D_s}$$

$$= \frac{1}{D_s} \cdot \frac{S_m}{m} + S_{cnd} \cdot \left(\frac{D_r}{r} \cdot \frac{1}{N_S} + \frac{1}{r} \cdot \frac{1}{D_s}\right) \qquad (1)$$

Because the same computing environment, we set the parameters, m, D_r, r, N_s, D_s, to the same value in all of the SRNN algorithms. Therefore, we can combine part of the same parameters into α, β and we can get the formula (2),

$$S(q) = S_m \cdot \frac{1}{D_s} \cdot \alpha + S_{cnd} \cdot \beta \qquad (2)$$

Theorem 1

$S_B(q) < S_L(q) < S_E(q)$, where S_B, S_L, S_E are the cost of Basic-SRNN, Lazy-SRNN and Eager-SRNN algorithms respectively.

Proof

In Basic-SRNN, formula (2) are directly used because it is not optimized,

$$S_B(q) = S_m \cdot \frac{1}{D_{B_S}} \cdot \alpha + S_{B_{cnd}} \cdot \beta \qquad (3)$$

Similarly, in Lazy-SRNN, And in Eager-SRNN.

Using the half-plan pruning method in Lazy-SRNN can cause the processing efficiency of D_{B_S} in Map step is greatly increased. And Eager-SRNN not only uses half-plane pruning, but also saves

intermediate results in the distributed cache rather than in the disk. Thus, D_{E_S} is the biggest of them. That is, $D_{E_S} > D_{L_S} > D_{B_S}$. $S_{B_{cnd}}$ is the biggest because all of the neighbors around q are the candidate set in the Basic-SRNN. Lazy-SRNN may get additional candidate objects which are not a reverse nearest neighbor in filter step of SRNN, while Eager-SRNN using global pruned information to filter the unpromising data. Therefore, we find $S_{E_{cnd}} < S_{L_{cnd}} < S_{B_{cnd}}$. In addition, candidate result data set is much less than the original input data set in SRNN, Thus, we can get,

$$S_m \cdot \frac{1}{D_{B_S}} \cdot \alpha + S_{B_{cnd}} \cdot \beta < S_m \cdot \frac{1}{D_{L_S}} \cdot \alpha + S_{L_{cnd}} \cdot \beta < S_m \cdot \frac{1}{D_{E_S}} \cdot \alpha + S_{E_{cnd}} \cdot \beta \quad (4)$$

That is, $S_B(q) < S_L(q) < S_E(q)$.

Experimental Study

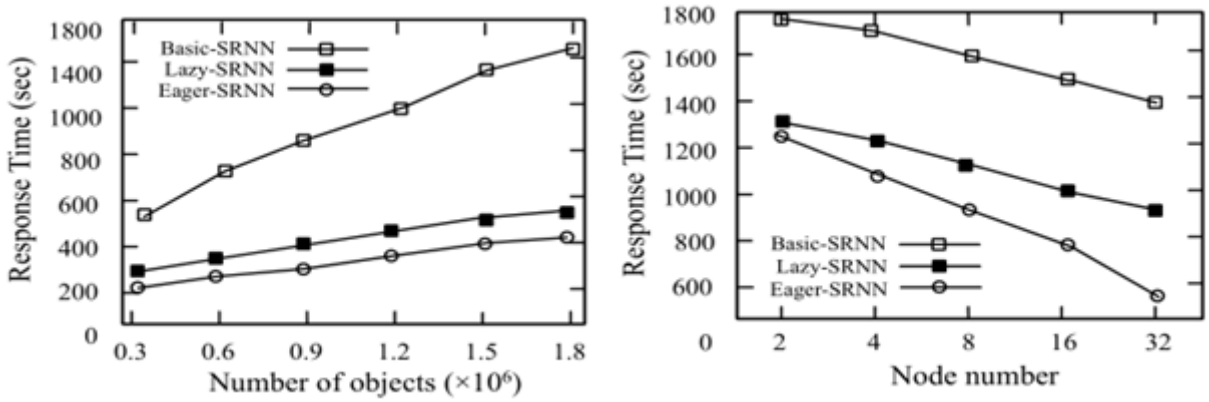
In this section, we experimentally evaluate the performance of SRNN for one type of Basic and two types of optimization strategies. And we compare SRNN against the state-of-the-art algorithm for RNN queries TPL [4] in non-distributed environment.

We set up a cluster of 32 commodity PCs in 100M high speed Gigabit Ethernet, each of which has a Dual Core AMD 2.00 GHz CPU,73GB of SCSI hard drive, 8GB memory, network and Ubuntu10.10 server OS. We use Cloudera Hadoop 0.20.2 and compile the source codes under JDK

1.6. One TaskTracker and DataNode daemon run on each slave. A single NameNode and JobTracker run as the master.DFS chunk size is 64 MB.

We used the real-world data set (RDS) in our experiments is the taxi GPS data with traffic conditions of Shenzhen City [11], it includes approximately 180,000,000 data points, and the raw data was about ten gigabytes before decompression. We used a subset of 1,800,000 data points in our experiments, where each data point represents a breaking behavior of tracking vehicle, and each data point contains seven dimensionalities.

In this experiment, we compare the native Basic-SRNN and optimized Lazy-SRNN and Eager-SRNN algorithms. Figure 2 shows the performance of change data size and cluster size to evaluate SRNN query. From that we can find the performance of Lazy-SRNN and Eager-SRNN is improved significantly, because SRNN algorithms have half-space pruning filter some data points and MapReduce parallel to improve the scalability of query processing. And in Figure 2(a), we can find Basic-SRNN may go through some nodes that can be pruned and subsequently, access most of the spatial space. When the Lazy-SRNN and Eager-SRNN increase the shuffle phase of reducing task, the performance is improved greatly as shown in the Figure 2(a). When the objects number is 1.8 × 10 6, the execution time of Lazy-SRNN is 5 times less than Basic-SRNN.



(a) Data point cardinality (b) Scalability

Figure 2. Comparison of SRNN

In Figure 2(b), the running time of SRNN decreases for enlarging the parallelism of the system as the number of cluster node increases. The TPL and Voronoi algorithms are not as good as the

Lazy-SRNN algorithms at performance and scale. And in running time, the Lazy-SRNN and Eager-SRNN of optimal strategies are both better than Basic-SRNN. When the node number is 16, the execution time of Lazy-SRNN is 1.9 times less than Basic-SRNN.

Conclusion

This work studies the RNN query distributed MapReduce. Our optimal Eager-SRNN delivers performance is better than baseline method, in order to enhance the scalability and performance of large scale Reverse Nearest Neighbor query. We also have conducted extensive experiments over both synthetic and real world spatial data to evaluate these algorithms. Finally, we find an interesting future research direction is to develop algorithms for efficiently answering Bichromatic Reverse Nearest Neighbors (BRNN) and Continuously Reverse Nearest Neighbor queries (CRNN) for moving points.

Acknowledgement

This work is supported by the Project of College Students' Innovative and Entrepreneurial Training Program (201411258048 and 201411258010), the general program of Liaoning Provincial Department of Education Scientific Research (L2014492 and L2014283), the Twelve-Five Teaching-reform Planning Project of LiaoNing Province of China (JG14DB037), and the Teaching-reform Project of Dalian University of China (2013122G1 and 2013123G1).

References

[1] C. Ji, Y. Li, W. Qiu, Y. Jin, Y. Xu, U. Awada, K. Li, and W. Qu. Big data processing:Big challenges and opportunities. Journal of Interconnection Networks, 13(03n04), 2012.
[2] Y. Shao, J. Xie, Y. Li, S. Gao and C. Ji. Efficient Distributed RNN Query Processing with Caching. In Applied Mechanics and Materials, pages 5352–5355. Trans Tech Publ, 2014.
[3] F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In ACM SIGMOD Record,volume 29, pages 201–212. ACM, 2000.
[4] Y. Tao, D. Papadias, and X. Lian. Reverse knn search in arbitrary dimensionality. In Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30, VLDB ’04, pages 744–755. VLDB Endowment,2004.
[5] C. Ji, Z. Li, W. Qu, Y. Xu and Y. Li. Scalable nearest neighbor query processing based on Inverted Grid Index. In Journal of Network and Computer Applications, , volume 44, pages 172-182. 2014.
[6] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. Communications of the ACM,51(1):107–113, 2008.
[7] C. Ji, H. Hu, Y. Xu, Y. Li, and W. Qu. Efficient multi-dimensional spatial rknn query processing with mapreduce.In ChinaGrid Annual Conference (ChinaGrid), 2013 8th, pages 63–68. IEEE, 2013.
[8] L. Ding, G. Wang, J. Xin, X. Wang, S. Huang, and R. Zhang. Commapreduce: an improvement of mapreduce with lightweight communication mechanisms. Data & Knowledge Engineering, 88:224–247, 2013.
[9] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. Mapreduce online. In NSDI, volume 10, page 20, 2010.
[10] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari. S4: Distributed stream computing platform. In Data Mining Workshops (ICDMW), 2010 IEEE International Conference on, pages 170–177. IEEE, 2010.
[11] Comap. the consortium for mathematics and its applications. http://www.comap.com