On Massive Spatial Data Retrieval Based on Spark

Xiaolan Xie(✉), Zhuang Xiong, Xin Hu, Guoqing Zhou, and Jinsheng Ni

Institute of Information Science and Engineering, Guilin University of Technology, Guilin, China
xie_xiao_lan@foxmail.com

Abstract. In order to search more efficiently for rapidly growing spatial data, cloud quad tree and R-tree is adopted in spatial index for the non-relational databases of cloud HBase, by which data can be retrieved successfully. By comparison of retrieval efficiency between cloud quad tree and R-tree, that how different parameters acted is tested on data index and retrieval efficiency and put forward an relatively more reasonable solution. Subsequently, we verify the validity of index calculation when there is a giant one.

Keywords: Spatial data · Spark · Quad tree · R tree · Index · Cloud computing

1 Introduction

With the growing development of computer technology, Geographic Information System (GIS) have been developing rapidly, with the expansion and augment of its application and demand. For faster processing and better spatial index, cloud computing is applied to retrieve spatial data prove to be a leading subject of resolving massive spatial data and geographic distribution [1].

R-tree (Rectangle Tree) is a data structure which is widely used in spatial retrieval. The depth of the tree is relatively smaller than the quad-tree, therefore the query efficiency is much higher. R-tree as an index can achieve a great faster speed in less changing spatial data. Besides, the controllable section for the user to control is much less and therefore it is more convenient to use. However, the add or delete operation on R-tree, which has already been generated, will decrease the quality of R-tree index, while it will consume a huge amount of computing if the index is repeatedly rebuilt (especially at present a lot of mutation of R tree, such as R+ tree, R* tree, are at the expense of insert and modification to improve the performance of query). As for the demand for this type, the quad-tree as represented in segmentation rule is a better choice. In this paper, we achieve a quad-tree and R-tree algorithm for parallel query based on Spark, to lay the foundation for the massive spatial data query and processing [2].

Y. Chen et al. (Eds.): WAIM 2014, LNCS 8597, pp. 200–208, 2014.
DOI: 10.1007/978-3-319-11538-2_19

2 Quad-Tree Index Algorithm on Hbase

The other paper entitled On Massive Spatial Data Cloud Storage and Quad-tree Index Based on the Hbase design a linear quad-tree spatial data index structure based on Hbase, but MapReduce is a framework of batch to process massive data, and it is not appropriate for the query operation. As for this type of operation, the memory computing framework Spark is a better choice.

Spark index quad-tree is divided into three steps: preparation in advance, computing, and consolidation. The Hbase query interval data will be read into a RDD block before Spark index, and then the data block will be calculated. Compared to calculation and operation on the Region of MapReduce, the same operation in memory RDD block is much faster and more stable. The index procedure for Spark on quad-tree cloud is as following program flow chart [3] (Fig. 1):

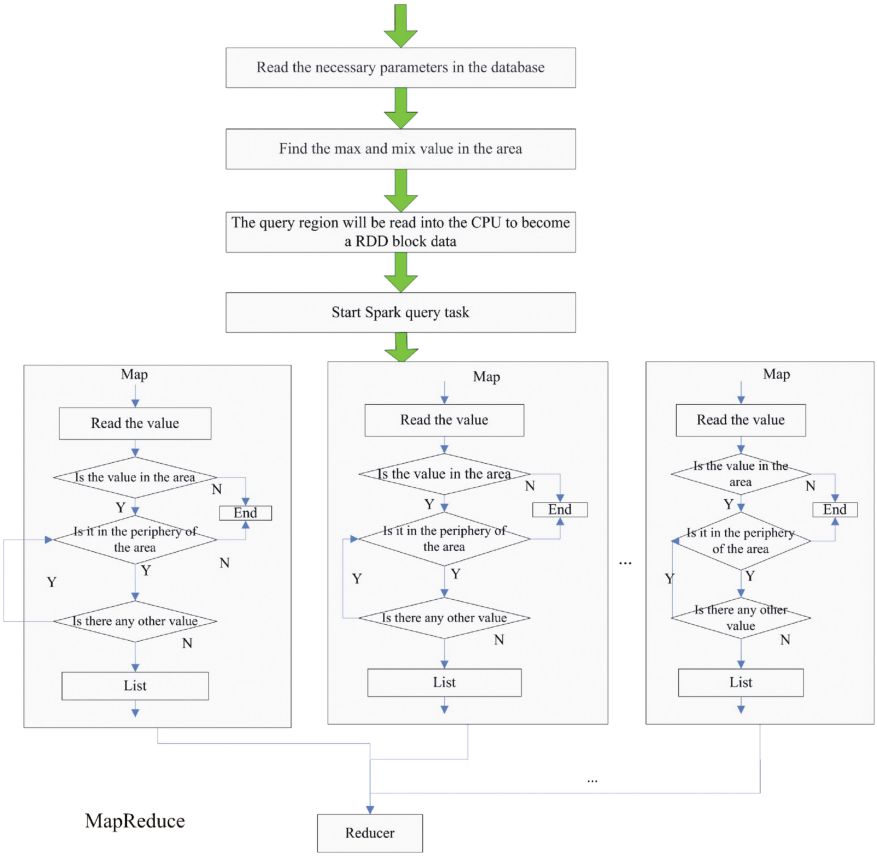

Fig. 1. Program flow chart for Spark on quad-tree cloud

Map operation output of regional spatial data may have duplicate values, therefore, Reduce operation must delete the duplicate values, and ultimately the same spatial data only output once. The nonexistent duplicate value after the Reduce operation of spatial data sets is the query results.

3 R Tree and Variants

Traditional sort tree, B tree appeared to be inadequate in the spatial data index. Because the traditional index tree is mostly linear index tree, the spatial data are basically more than two-dimensional spatial data, which does not have a very clear linear structure. In order to solve this problem, Guttman proposed R-tree, and then the advantages of dynamic balance is extended to the multidimensional space [3]. As a result the sort tree can be used in lots of new area, such as spatial data.

The directory node of the R tree is mainly composed of Mbr and sub node array, where Mbr is the smallest rectangle containing the node, and the rectangle can accommodate all the child nodes of the node. Data node of the R tree is mainly composed of Mbr and spatial data array, R tree by itself generally does not preserve spatial object itself, but to save space in the location of the database object. The biggest problem of R-tree is the brother node Mbr overlap. If the query region contains exactly several nodes of coincident area, the index of these nodes under the same root of the subtree must search. In order to minimize overlap, domestic and foreign scholars have proposed many R-tree variants, such as R+ trees, R* tree, X trees [4]. These improvements have advantages and disadvantages, and R * tree is generally considered as the most efficient R-tree variants.

4 R-Tree Index Algorithm on Spark

4.1 Traditional R-Tree Index Algorithm

Traditional R tree and its various variants of index algorithms can be roughly divided into three steps [3–5]:

- If the root node R overlaps with the query area S, and R is a directory node, it is used as an ordinary node T, and then carry out the second step. If the root node R overlaps with the query area S, and R is the data node, and then it is treated as a ordinary node T, and then carry out the third step.
- If T is not a leaf node, every child node of T will be checked to determine whether the child node overlaps the S. For all overlapping sub nodes, the index algorithm will be used to search for subtree under the root of sub nodes.
- If T is a leaf node, every child node of T will be checked to determine whether the child node overlaps the S. If the data Doverlaps S, the data D is one of the search condition in accordance with the data.

The main steps are highly parallel computing, and do not affect each other. Therefore, the algorithm will be directly converted into cloud algorithm, which is

theoretically feasible. But it is found that in actual operation the direct conversion of these steps to the cloud operation cannot bring too much performance as Spark comparison algorithm running intersection is relatively more rapid, and the distribution and aggregation of data blocks is relatively slow. Reduction operation is frequently used and this will consume a lot of network bandwidth and transmission time, especially during the iteration, the bandwidth will become a significant bottleneck. Therefore, we can consider the query K layer subtree in one time of iteration, Reduce operation will be processed after each K layer query, then re-allocate computing tasks.

4.2 Cloud R-Tree Index Algorithm and Optimizing

Cloud R tree query algorithm is also divided into three steps, which can be subdivided into the following steps [5–8]:

(1) If the root node R intersect with the query region S, which is a directory node, put the root node into Dir List, and carry out the second step. If the root node R intersect with the query region S, which is a data node, then put the root into DataNode List, and carry out the third step.

(2) Recursive query is used for each node of Dir List until all the nodes are up to leaf nodes.

- Convert Dirlist to RDD data block, denoted Rd, is the value of k is default to 0;
- Perform the Map operation of Rd data block, if Rdi intersects with S, then output all of its nonempty nodes as an array, k value plus 1;
- If k < K and the output is not empty and the child node of output node is not a leaf node, continue to process 2.2 until k = K or the output is empty or the child node of output node to the leaf node;
- Perform the Reduce operation of the Ra data block RDD after the Map operation, combine all the outputs with a large array, and convert the array into List, denote Li.
- If the elements of Li are Li leaf node, perform the third step, otherwise, make Li as the new DirList, continue to process 2.1 until Li is a leaf node.

(3) Determine whether the data contained in DataNode List overlap S, output the overlapping data.

- Convert The Data Node List into a RDD data blocks, denoted by Rn;
- Perform the Map operation on data block Rn. If Rni intersects with S, then output all if its data to an array.
- Perform the Reduce operation on data block Ra after the Map operation, combine all the outputs into a large array, and convert the array into List, denoted Ld.
- Convert Ld into the RDD data block, denoted by Re;
- Perform Map operation on Re. If Re (i) intersects with S, Re [i] data is one of the index criteria that meet the requirement;
- Perform the collect operation on the Ra data block after the Map operation, and obtain the final result.

The value of K is determined by the number of the spatial data and the order of the R tree itself. The smaller the order is, the deeper depth of the tree. While the data for

each node is relatively fast, K can be set to a larger value. Meanwhile, if the network bandwidth is small, Reducing operating costs is relatively large. It can increase the K value in order to reduce the Reduce operation frequency.

5 Simulation

5.1 Experimental Platform

Hardware: Master Node: 3.40 GHz i5-3570 CPU 4G RAM;
Slave nodes: four 3.30 GHz i3-3220 CPU 4G RAM;
Platform: Fedora18 x86_64 Hadoop-1.0.3 HBase-0.94.2 mesos-0.12.1 spark-0.7.3;
Java environment: jdk-7u40;
Development and debugging tools: Eclipse J2EE Galleo;

Experimental data: about 90 % of them with text describing properties, 50 % with image description attributes, 30 % with audio description attributes, 10 % with video description attribute.

5.2 Comparison of Performance Between Spark and Hadoop Retrieval

In the Spark and Hadoop clusters are retrieval operations on spatial data of different sizes of space, record retrieval time as shown in Fig. 2 (unit: Ms):

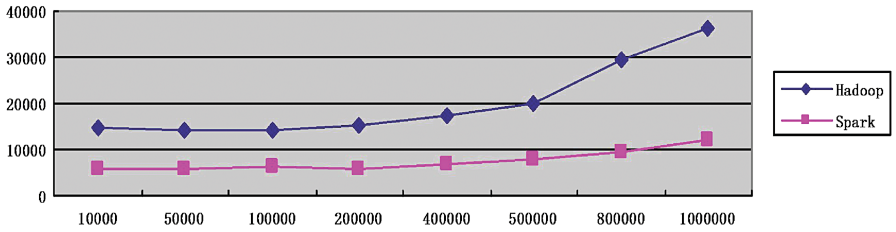

Fig. 2. Comparison of different regional spatial index time

It can be seen from the table that the efficiency of Spark query is significantly higher than conventional MapReduce. On one hand, the code of Spark is more compact and efficient, on the other hand, the computational model and HDFS coupling of MapReduce is too high, and most of the convergence of computing operations are required to read and write HDFS, while Spark are to be treated as a block of memory to process the data and process results, it not only improves the data processing speed and fault tolerance, but also reduces the procedures for hard disk and network IO requirements.

5.3 The Performance of Spark Spatial Data Retrieval

Using the Collect method to the RDD data block instead of the Reduce operation, the Spark cluster on the space point different size of data operation space retrieval operation, retrieval time record retrieval time and with regional data were compared, as shown in Fig. 3 (unit: Ms).

It is can be seen from the Table 2 that after the elimination of the step. Reduce operation of the spatial point cloud quad-tree query operation, the index is increased by nearly 10 times compared to Reduce operation. To improve the index efficiency, operations like space plotting should be processed by using as many as point data.

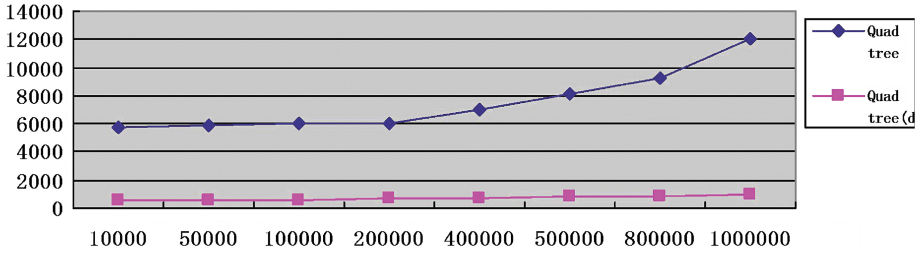

Fig. 3. Different types of spatial data index time

5.4 Cloud Quad Tree for the Uneven Distribution of Spatial Data Processing

To test the cloud quad tree unevenness of the distribution of the data processing, Java program is employed to distribute 50 percent of the data to around 10 % space with equality, and the other half of the data is still evenly distributed in the entire space. The test is divided into two steps: intersection with the search range and intensive space; or vise versa. The index time is recorded as shown in Table 1 (unit: ms, OM: table representative of a Java out of memory error):

Table 1. Non uniform spatial data index time

Test	The index range and space intensive intersection				The index range and space -intensive disjoint			
Volume data sheet	100,000	200,000	400,000	500,000	100,000	200,000	400,000	500,000
Level = 6	17809	18347	OM	OM	6065	6015	7276	8125
Level = 8	10250	10544	14628	17244	6102	6003	7256	8044
Level = 10	7125	7049	7536	8247	6097	6115	7179	8062

When Level is small, the imbalance data has had a great impact on the quad-tree. When the search range and dense space intersects, the level is too small to trigger the out of memory error. Because the division of cloud quad-tree space is completed before

the data is inserted, data amount of local area is increasing greatly, which has a particular similar effect on cloud quad-tree and the time under large global data amount. A large amount of individual data item for key causes a longer time to judge the space data communication when searching the data, and thereby it leads to the bottleneck for the inquiry. However, the bottleneck of this local query performance only appears in the query region which contains or intersects data dense areas, but it does not have any impact on query performance of sparse region. The solution for this problem is same with the problem of global intensive data, which is to increase the Level depth of cloud quad tree. When Level in Table 3 is big the increasing data of local data does not cause too much reduction in query efficiency, which also illustrates this point.

5.5 Comparison of Reading Time Between Quad-Tree and R-Tree

Creating quad-tree and R -tree index in HBase for different sizes of spatial data, and then using the Spark to read the index and the establishment of a tree query structure, time consumption is shown in Table 2 (unit: ms):

Table 2. Time setting for quad-tree and R-tree

Amount of data	10,000	50,000	100,000	200,000	400,000	500,000	800,000	1,000,000
Quad-tree	80	82	81	95	102	134	183	227
R Tree	1147	1453	2139	4412	6435	7946	9532	12453

In order to create the directory node, R-tree needs to repeatedly read HBase database, causing the result that the index time of spark R-tree is much greater than R-tree. In addition, all the index of R-tree must be entirely read to CPU and creates a tree structure. Compared to quad tree which may contain parts of the data read into the memory space, the flexibility of the R-tree seemed to be insufficient. Therefore, when compared to the query efficiency of the quad tree and R tree, the query time will only be calculated, without considering the creation of R-tree and cloud quad-tree time required to read from HBase. R-tree is mostly used for static data index, while, when retrieval operations are very frequent, you can save time by using the generated R tree shared by multiple retrieval operations.

5.6 Comparison of the Retrieval Time of R Tree and Space Quad Tree under Different Parameters

The value M of R tree and value K of the index program are set to different values, and perform retrieval operations for spatial data of different sizes in the Spark cluster. The comparison of time between record index time and cloud quad tree index time are as shown in Table 3 (unit: Ms):

Table 3. Comparison of index time between quad tree and R tree

The amount of data	10 thousands	50 thousands	100 thousand	200 thousand	400 thousand	500 thousand	800 thousand	1 million
Quad tree(point)	579	606	593	632	759	820	883	965
Quad tree(region)	5786	5946	6082	5986	7043	8146	9237	12103
M = 100 K = 4	1090	1786	4450	6670	8753	9548	23436	26690
M = 100 K = 8	1850	1927	4197	6375	8032	8113	13042	17890
M = 500 K = 4	1136	1534	1942	5236	7526	7845	8932	11279
M = 500 K = 8	1627	2179	3045	5124	7372	6239	9768	15367

It is shown that, optimized space quad-tree points query index is much faster than other index methods. While the value M of R tree and value K of the index program are taken to the appropriate time, the query efficiency of spatial data is much higher than the quad tree. But if both of them do not match well, the query efficiency of R-tree will be greatly affected, especially in a large amount of data. The choice of smaller K and value M will cause frequent transfer of Reduce, which could greatly decrease the efficiency of R-tree index. When value M is larger, the depth of R tree is smaller, while if value K is set to a larger value, unbalanced load will be caused. Meanwhile, M value will be bound to the use of memory (out of memory error occurs when the value of M in the table above), and the larger value M will make the MBR of R-tree node overlap rate increase, so the value of M and K must be adjusted according to the dynamic demand: when value M increases, K values should be reduced accordingly.

6 Summary

Due to the quality of cloud storage by columns, smaller computing needs larger IO, data arranged according to Key, etc., there are many differences between the cloud spatial indexing and retrieval with single selection. As changes in less space area data, query efficiency is slightly higher than R tree spatial quad tree, and cache memory sharing technology to reduce the generation time R-tree, and after each data changes, you must re-build the R-tree index. If the data changes greatly, you can use the space for better overall performance quad-tree. For processing point data, the algorithm efficiency is much higher than the R-tree following the space-optimized quad-tree search. Therefore, it's better to use quad-tree spatial point data retrieval in the cloud.

Acknowledgments. This research work was supported by National High Technology Research and Development Program 863 under Grant No. 2013AA12A402, Natural Science Foundation of Guangxi Provincial under Grant No. 2013GXNSFAA019349.

References

1. Zhu, Q., Zho, Y.: Distributed spatial data storage object. Geom. Inf. Sci. Wuhan Univ. **31**(5), 391–394 (2006)
2. He, R.Y., Li, Q.J., Fu, W.J.: Guide to Developing and Application With Oracle Spatial Database, pp.1–48. Surveying and Mapping Press (2008)

3. Hu, J.: Research and implementation of parallel clustering algorithm in cluster environment. Master's degree thesis of East China Normal University, pp.15–29 (2012)
4. He, X.Y., Min, H.Q.: Hilbert R- tree spatial index algorithm based on Clustering. Comput. Eng. **35**(9), 40–42 (2009)
5. Zhang, Z.B., Wang, Y.Z., Li, H.: Research on the cost model for spatial query based on R-Tree. Micro Comput. Syst. **24**(6), 1017–1020 (2003)
6. Cary, A., Sun, Z., Hristidis, V., Rishe, N.: Experiences on processing spatial data with MapReduce. Sci. Stat. Database Manage. **2009**(6), 302–319 (2009)
7. Wang, L., Zhang, H.H., Li, K.S., Ju, H.B.: Region matching algorithm for DDM based on dynamic R-tree. Comput. Eng. **34**(3), 56–58 (2008)
8. Apache. Spark Handbook (2013). http://spark.incubator.apache.org/docs/0.7.3/