International Journal of Grid and Distributed Computing
Vol. 9, No. 1 (2016), pp.1-12
http://dx.doi.org/10.14257/ijgdc.2016.9.1.01

Research on Multi-Dimensional Index in Cloud Computing System

Qinghong Liu [1] and Hongyan Zhang[2]

[1,2]*Jilin Province Economic Management Cadre College, Changchun 130012, China*
[1]*384243571@qq.com*, [2]*304821123@qq.com*

Abstract

Existing works mainly focus on indexes in a single server or server-client structure. Unfortunately, such works fail to provide efficiency since performance bottleneck is introduced. This paper designs a two-layered index structure to prune search space among computing nodes for query processing. The index reduces the number of involved computing nodes while query processing, and improves I/O efficiency inside a single server. The initiation method and maintenance methods are proposed for the index, together with optimization strategies for improving query throuphput. This thesis designs the point query algorithm, the range query algorithm and the kNN query algorithm for cloud systems, including distributed algorithms among computing nodes, and optimization strategies inside a single computing node.

Keywords*: multi-dimensional index, cloud computing, spatial database, query processing*

1. Introduction

Due to the shortage of scalability and reliability, traditional database technologies can't be applied directly on cloud computing platform. So it's challenging to build database service in the cloud computing system. Meanwhile, plentiful applications in the cloud computing system depend on database services in the back end. Such applications rely also on the performance and quality of database services in such systems [1-3].

Online analysis processing and online transaction processing are two main workloads to the database system. The current cloud computing system performs mainly on-line data analysis of multi-scale data collections and it has created system like MapReduce, HadoopDB and Scope [4-5]. Such kind of systems decomposes data analysis tasks to many subtasks which can be treated in a parallel manner, with numerous computing nodes to simultaneously process subtasks. Finally the system integrates intermediate results generated by subtasks [6-8]. The optimal objective of in-line data analysis is to reduce consumed time for data analysis. Different from that, online transaction processing accepts huge concurrent data read-write requests [9-10]. It can locate effectively all data involved with such requests to improve the effectiveness of online transaction processing systems and obtain higher throughput of them. Traditional online transaction processing systems make use of indexing technology to position data in the external storage memory, cutting down I/O overheads of queries. To set up online transaction processing systems in cloud computing system, the effective indexing technique is required as to locate needed data among those computing nodes used by the system [11-13].

RTN will calculate the node organization system into Content Addressable Network structure, in which each computing node is responsible for receiving the query routing system. In order to improve the efficiency of local queries, computing nodes using R tree index data. Each local data release index called the global index, used to locate the query data needed in the system. In order to reduce the number of global index, computing

ISSN: 2005-4262 IJGDC

nodes using a local R tree node as a global index, a global index system is the summary of the nodes in the local data. Therefore, the design of the index structure named RTN (R-Tree Network). Used the R tree node as a global index can reduce global index storage and maintenance cost, and global index in the R tree node can effectively locate the nodes in data location.

2. RTN Index

2.1. Structure of the System

RTN index works on sharing-nothing distributed clusters. The data collections are divided into data blocks and distributed among each computing node in the system. As seen from Figure 1, each computing node N_i N in the system plays two roles, which is respectively storage node N_{si} and index node N_{oi}. N joins in the distributed storage system and is responsible for storing partial data of the system. To quicken up local multi-dimensional data inquiry response, N_{si} uses R tree to index local data. N_{si} interacts with N_{oi} to release global index. N_{oi} is one node among the structured peer-to-peer (P2P) network topology CAN employed by the system. It is responsible for maintaining one spatial partition of the whole multi-dimensional data space. N_{si} adapts according to the current workload to choose some nodes from the local R tree as global indices, which are then distributed by N_{si} to the system through the P2P network topological interface provided by N_{oi}, in the form of IP. There, ip is the IP address of index owner N_i; port is network port used by N_i; addr is the memory address of R tree node to which the global index corresponds in the external memory; mbr is the smallest multi-dimensional area of R tree node to which the global index corresponds. When N_{oi} receives the global index releasing request from N_{si}, N_{oi} maps relative R tree node to the index node in CAN; then by CAN's routing protocol, such request is sent to the relevant index node. N_{oi} is responsible for the maintenance of global index in the system. When it gets the global index releasing request from the system, N_{oi} firstly utilizes RTN's mapping mechanism to examine if it's the recipient of the global index. If N_{oi} is the receiver, the global index is saved in its internal memory and used to boost the inquiry response. In RTN's index structure, the global index is composed of nodes chosen from the local R tree and distributed among index nodes in the system.

2.2. Mapping Mechanism of Global Index

RTN uses C^2 P2P network structure to maintain the global index of multi-dimensional data in the system. C^2 C P2P network structure improves search quality on the basis of CAN, by adding some logical links to neighbors in each dimension. And it constrains the searched routing message cost within O(logN), where N means the quantity of nodes in the system. RTN has a mapping mechanism, which maps R tree nodes to nodes of many C^2 P2P networks and assigns those nodes as the recipient of the global index. For the given R tree node n, RTN calculates firstly the center c_n and radius r_n and uses them during the mapping. If n's radius r_n is below the predefined threshold $r_{\max}$ by the system, n will be mapped to one index node; otherwise, n will be mapped to many index nodes. RTN uses the global index mapping method in the following steps. For the given R tree node n, RTN maps n to one index node N_c, by which the division part of the maintained

multi-dimensional data space contains the center c_n of R tree node n. Then, N_c compares n's radius r_n with the preset threshold $r_{\max}$ by the system. If r_n is smaller than $r_{\max}$, N_c will send n to index nodes of n overlapping with all multi-dimensional space areas. During the mapping, RTN utilizes n's center because in R tree nodes close to the center, similar data are more probably contained, *i.e*. the data accessed by the same inquiry.

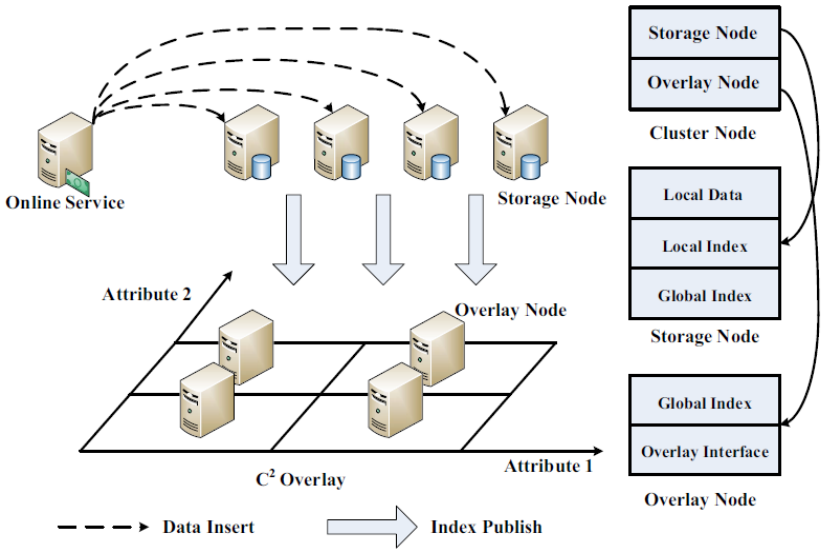

Figure 1. RTN System Architecture

Parameter $r_{\max}$ is very important to the search space of query in the system. The increasing $r_{\max}$ may lead to more inquiries in the system accessing more index nodes. Figure 2. is an example of different query search spaces for different values of $r_{\max}$. In one two-dimensional C^2 P2P network, when $r_{\max}$ is made R_1 and R_2, search spaces of node query are round areas of which the radius is respectively R_1 and R_2, as seen from the picture. If $r_{\max}$ is increased from R_1 to R_2, the query needs accessing more search nodes as to acquire all query-intersected global indices. In the system, $r_{\max}$ is adjustable. Bigger $r_{\max}$ degrades the inquiry effect because more index nodes will be accessed; on the other hand, smaller $r_{\max}$ brings about bigger costs for maintaining the global index, because more global indices have several backups in the system. In the RTN index used here, $r_{\max}$ is bigger than the radius of most R tree nodes in the system and its value can be set according to the sampling of R tree node size.

3. Query Processing

The paper made examples of multi-dimensional point query, range query and KNN query to introduce the method based on RTN for processing multi-dimensional inquiries. As mentioned before, query processing includes two stages. On the first stage, query obtains through the global index all memory nodes which may contain the inquiry results; on the second stage, query is routed to those storage nodes and searches locally R tree and get the result. In the paper, we chose the first stage to discuss further since the second stage is quite alike to the current concentrated system. Give multi-dimensional query Q. To ensure the completeness of query result, Q needs being routed to index nodes in all

multi-dimensional space areas which are overlapping with Q. We propose an effective algorithm to narrow the query searching space, which uses the center and radius of R tree node. Suppose the dimension of C^2 P2P network is d and give out the algorithm for RTN processing multi-dimensional queries and KNN queries. Figure 3- 4 shows the space of range query searching global indexes.

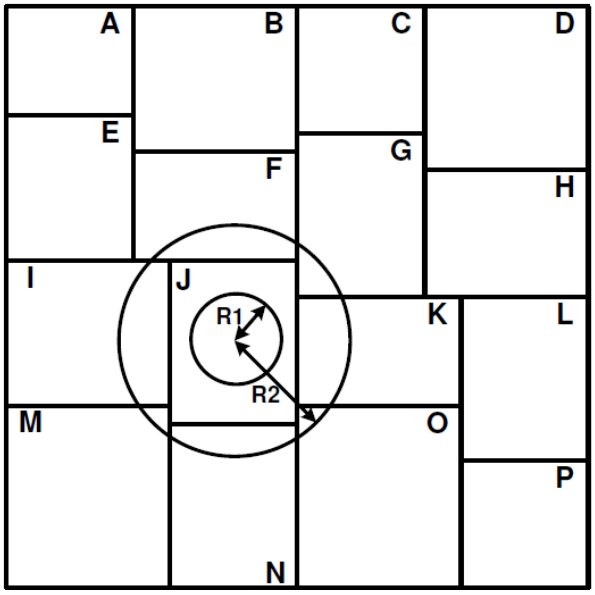

Figure 2. Different Search Space of an Point Query with Different $r_{\max}$ Values (R_1 and R_2)

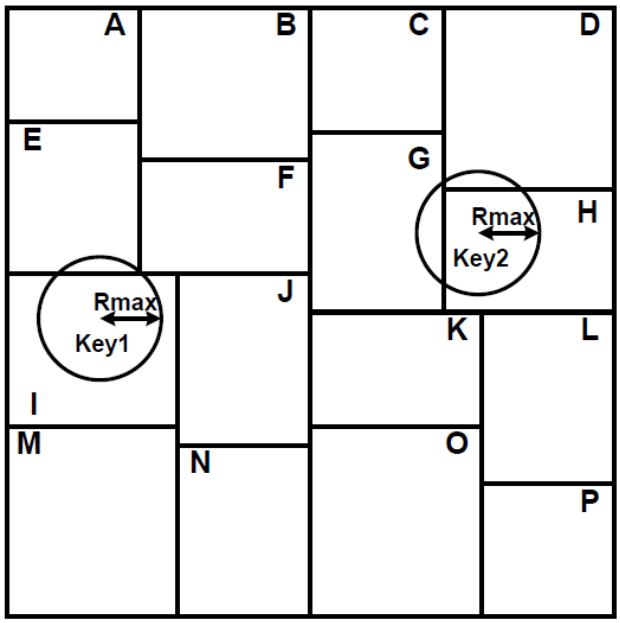

Figure 3. Search Space of Point Query

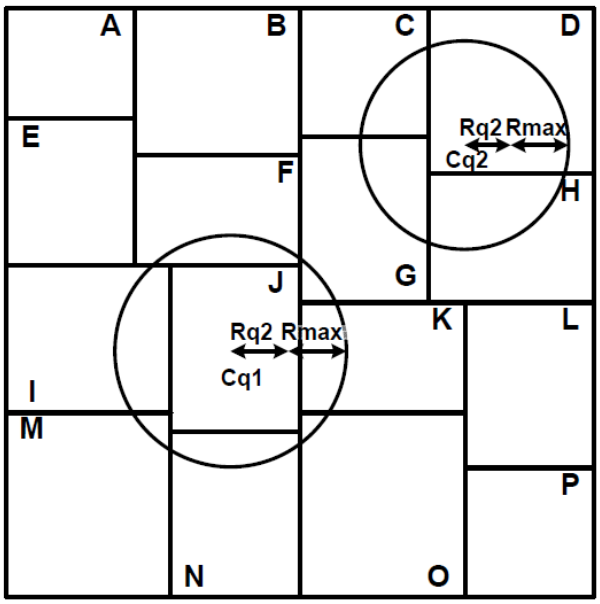

Figure 4. Search Space of Range Query

3.1 Range Query Processing

Given one multi-dimensional range query Q(range). The multi-dimensional range query processing firstly needs to find the global index of all multi-dimensional space areas intersecting with query range. This step is similar to multi-dimensional point query handling. Firstly, query is routed to one index node N_{ini}, the multi-dimensional space area in whose charge has the center c_r of query range; next, query is sent by N_{ini} to one index node in the global index searching space C. Q(range)'s sending pattern is identical to what's used in the point query processing. The searching space is split in dimensions. The split sub-areas are allocated to each neighbor. Index nodes which get sub-area and query further subdivide areas under their control according to the dimension; then send query to relevant adjacent nodes. In this period, all index nodes having received Q(range) search and inquire in the internal memory the global indexes intersected with query range. Next they return search results as for data acquisition. It shown in Algorithm 1

Algorithm 1. RANGE QUERY PROCESSING

Input: multidimensional range query Q (range)

Output: global index set S_i

1 $S_i = \varnothing$

2 $N_{init} = CAN.lookup(key)$

3 $C = generateSearchCircle(key, R_{\max} + r_r)$

4 For i=1 to d do

$C = R_0$

$N_1 = getNeighbor(N_{init}, R_1)$

$N_2 = getNeighbor(N_{init}, R_2)$

5 $S_i = N_{init}.globalIndex$

6 $For\ \forall I \in S_i$ do

$S_i = S_i - \{I\}$

7 Return S_i.

3.2 KNN Query Processing

Given kNN query Q(key,k). Query processing returns k multi-dimensional data closest to key. RTN handles multi-dimensional kNN queries with this idea: use one hypersphere as the initial query search area and obtain data from it. If the number of acquired data is less than k, it's necessary to enlarge query search space till enough many query results are returned, with k tuples which are the nearest to key as query results. Although the idea is similar to the algorithm for processing kNN in R tree, conventional R tree algorithms like MINDIST pruning can't work directly on RTN. That is because RTN is distributed as R tree nodes of the global index in each index node in the system. R tree algorithm is not applicable directly for single computing node. Given one multi-dimensional point key, which is mapped by RTN to the index node N_{ini}. In the system, it's a challenging job to find out the global index closest to key. N_{ini} can't find alone R tree node which suffices the requirement; instead, it has to work together with nearby nodes in each dimension as to get all possible global indices which meet with the requirement. This may generate costly communication expenditures. To obtain k data closest to key, the algorithm used here needs repeating for many times. During each cycle of operation, the algorithm requires communications among many index nodes. The applied algorithm excavates the parallelism of the system, which is simple and effective.

Algorithm 2. kNN QUERY PROCESSING

Input: multidimensional kNN query Q (key, K)

The Output: data set S_i

1 $S_i = \varnothing$

2 $N_{init} = CAN.lookup(key)$

3 $\delta = estimateRadius(k)$

4 $r = \delta$

5 while true do

$S_i = Search(key, r)$

$f \mid S_i \mid \geq k$ *then*

Else

$r = r + \delta$

4. Experiment Design and Discussion

This paper implements the RTN index system using Java language, used Amazon provide commercial cloud computing platform EC2 RTN index test the query performance and update performance. The experiment using the JDK version of Java 1.6.0.13, the compute nodes used small computing unit in EC2. Each computing node configuration is Xeon CPU and memory 1.7GB, frequency 1.7GHz, 160GB disk. Nodes are connected by 250Mbps LAN.

For testing, we use 32,64,96,128 computing nodes. The experiment uses two datasets. In the even data collection, the experiment produces 500000N multi-dimensional data objects by artificial synthesis, where N is the number of computing nodes in the system. Each object has 2-5 numerical attributes as multi-dimensional coordinates, which each is integer from [0, 10^9]. The whole dataset is divided to N_g portions. N_g is numerous times of N. Later on, data partitioning is distributed to each computing node. In that way, each computing node maintains several data partitioning and each has basically same data

volume. In the transportation data collection, the experiment uses data generator to produce 100,000N two-dimensional data objects. In the generating process, the map of one city is employed. N is the number of computing nodes used in the experiment. The data objects in traffic data collection are distributed divergently in the space. The system uses a similar approach as in the even dataset to separate traffic data set and arranges the acquired data division to computing nodes.

In each computing node, nodes utilizes R tree index to create R tree in local database. nodes utilizes R tree index to create R tree in local database. R tree's disk page size is 4KB. Each R tree node has about 125 index entries. The query distribution being tested here includes uniform distribution and inclined distribution. We use Zipfian distribution to generate query of inclined distribution and control the degree of such inquiry. Table1 lists parameter values used in the experiment and default values. I t is shown in Table 1.

Table 1. Experiment Parameters

Parameter	The value
Data dimension	2,3
The type of query	range query, kNN query
Query options	0.01%,0.05%,0.1%
Deviation factor	0,0.5,1.0
Whether to use the index adjustment	YES, NO
Whether to use the routing cache	YES, NO
The number of computing nodes	32,64, 96,128

4.1 Performance of Range Query

Multi-dimensional range query is one of the commonest queries in Web2.0 application. Multi-dimensional point query can be regarded as one special multi-dimensional range inquiry. The range length of each dimension is 0. KNN query processing in the system is completed through a series of multi-dimensional range queries. This part will test the performance of RTN processing multi-dimensional range queries. Here we use query selectivity to mean the proportion of query range to the whole data space. For instance, in the 3-dimensional space, the query selectivity of 0.05% indicates that the query length takes about 8% in each dimension. Figure 5-6 describes, in traffic dataset and uniform dataset, different query quality by computing nodes of different numbers processing the multi-dimensional range query at different query choice degrees. It's observed that with increasing number of computing nodes in the system, RTN's query throughput ascends in an approximately linear way. It suggests that RTN has good scalability, applicable for the cloud computing system consisted of tremendous computing nodes. With higher query selectivity, more global indices need to be obtained; meanwhile more local queries are implemented in storage nodes. Hence, the system's query throughput descends. Of them, throughput of traffic dataset is higher than uniform dataset, for the reason that the former is composed of 2-dimensional data and the latter is of 3-dimensional data. So the query in the latter requires more routing messages as to get more global indexes and local disk searches.

In reality, query is often of slant distribution, *i.e.* user often focuses data in one space area. Figure 7-8 outline the effects of query handling capacity of the tested skew distribution and different inclinations on query performance. In the experiment, the query for accessing data blocks is subject to zipfian distribution and the value of its skew factor is 0, 0.5 and 1. When it's 0, the query is distributed uniformly; when it's 1, 80% query will access 20% data and the query is very skew. Experimental results demonstrate that RTN's query throughput is the highest for evenly distributed data; however, the query performance degrades along with increasing skew degree of the query. As the global index of dynamic adjustment is applied, RTN can provide satisfactory query handling capacity even when the query is too skew.

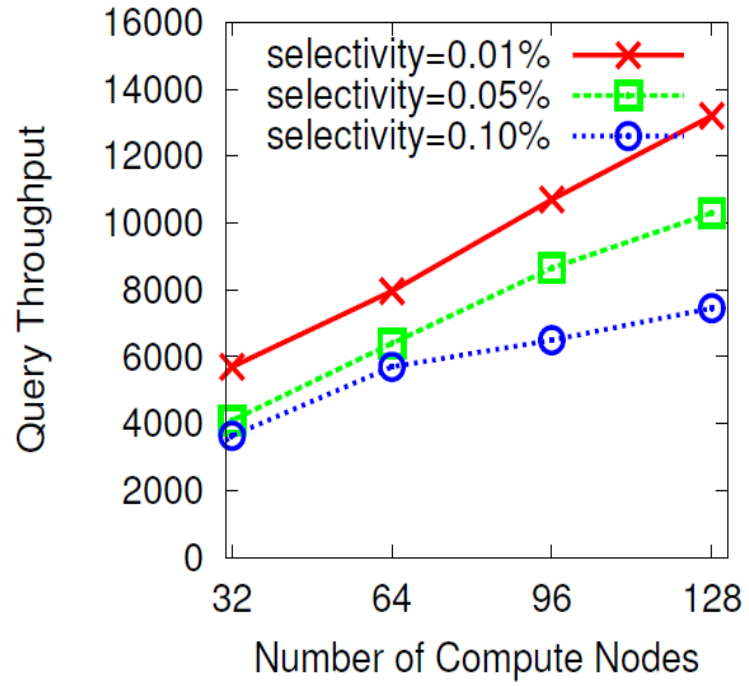

Figure 5. Effect of Selectivity (traffic Dataset)

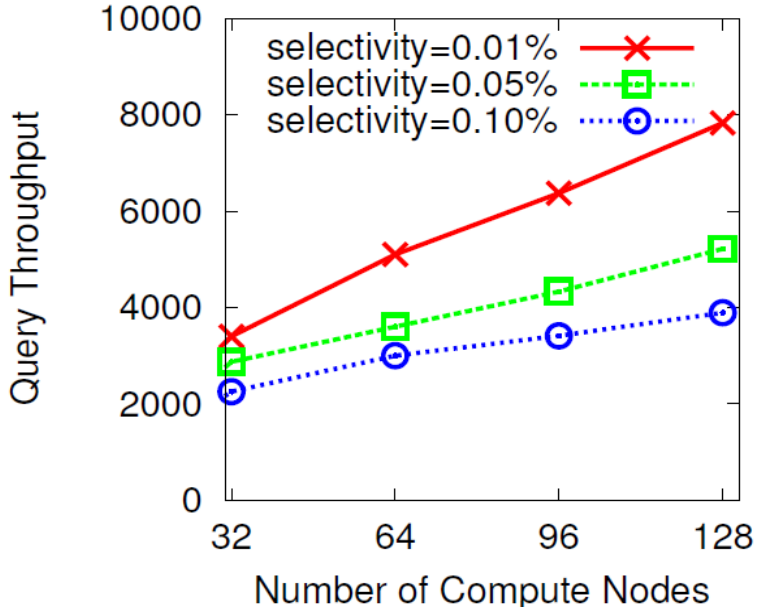

Figure 6. Effect of Selectivity (Uniform Dataset)

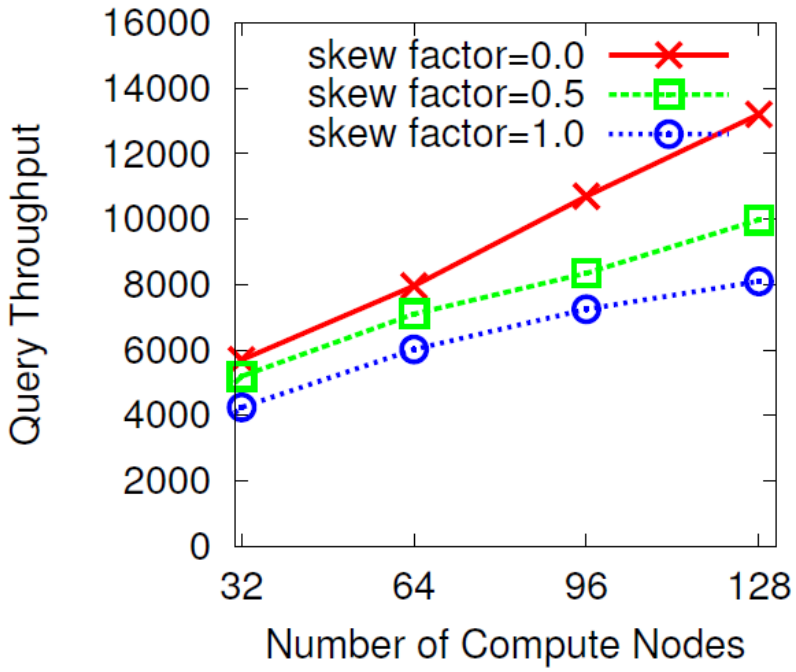

Figure 7. Effect of Skewness (Traffic Dataset)

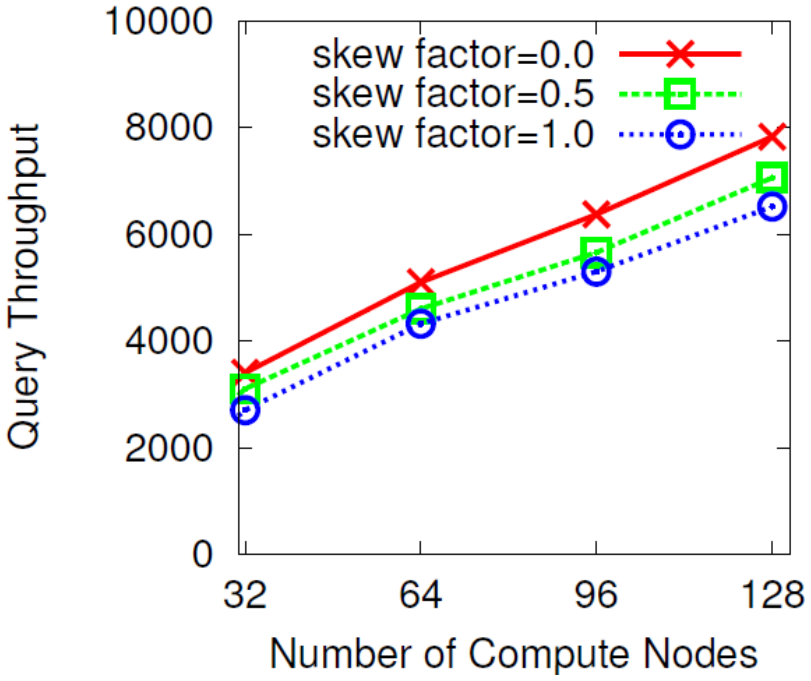

Figure 8. Effect of Skewness (Uniform Dataset)

4.2 Performance of KNN query

KNN query processing is more complicated than range query. The existing methods adopt generally progressive search and pruning strategies. The KNN query processing method used here is simple and useful. First of all, estimate the distance between queried target and the k*th* neighbor closest to it; then, with that estimation as the radius of searching space; next, start range query one by one, with the radius of the first range query as the initial estimated value; if the number of objects acquired in the last round is

below k, expand the search radius and start next range query; repeat this till the number of acquired objects is over k. Figure 9-10 portray the change of query throughput in the traffic dataset and even dataset when the parameter k is increased from 1 through 32. Figure 9. gives the query throughput of each computing node using 500K 3-dimensional objects. Figure 10. gives the query performance of each computing node using 100K 2-dimensional objects. Obviously, query throughput decreases with increasing k, because bigger k leads to bigger query searching space. As a result, more global indices will be accessed and further more storage nodes will implement local searches. In the traffic data collection, the query throughput reduces more slowly than in the even data collection. The reason is in the former dataset, data are more inclined; when k is increased, the query can access adequate query objects without enlarging considerably the searching space.

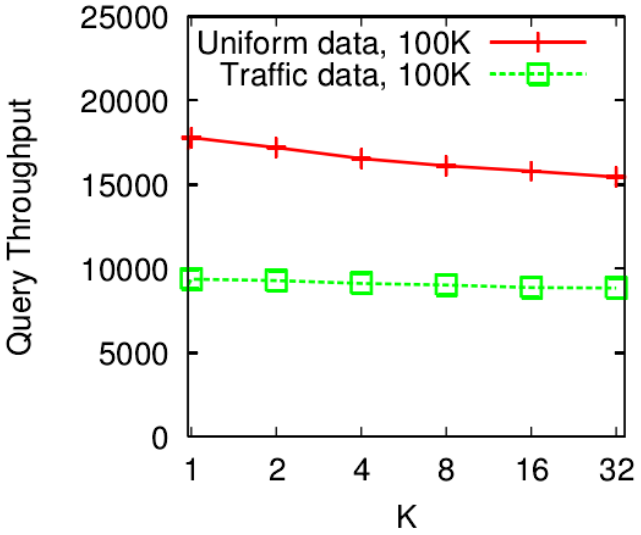

Figure 9. Performance of kNN Query(Traffic Dataset)

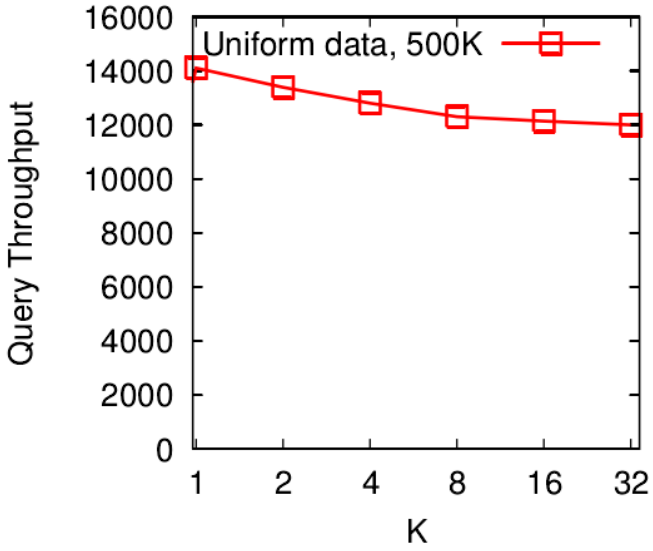

Figure 10. Performance of kNN Query(Uniform Dataset)

Figure 11-12 present the varied kNN query throughput with different computing nodes in the system. The computing nodes in the experiment increase from 32 to 128. Query parameter k is made 16. With more computing nodes, kNN query throughput increases linearly, meaning that kNN query processing algorithm has nice scalability. After

comparison of the performance between kNN query and the above range query, we learn that in the traffic data collection, range query achieved higher throughput; while in the uniform data collection, kNN query had higher throughput. In the traffic dataset, the query is more skew; kNN query generally requires several range queries in order to obtain enough query results. In the uniform data collection, kNN query generated search space is much smaller than that in the range query. In different data collections, the comparative results of performance are different between kNN query and range query.

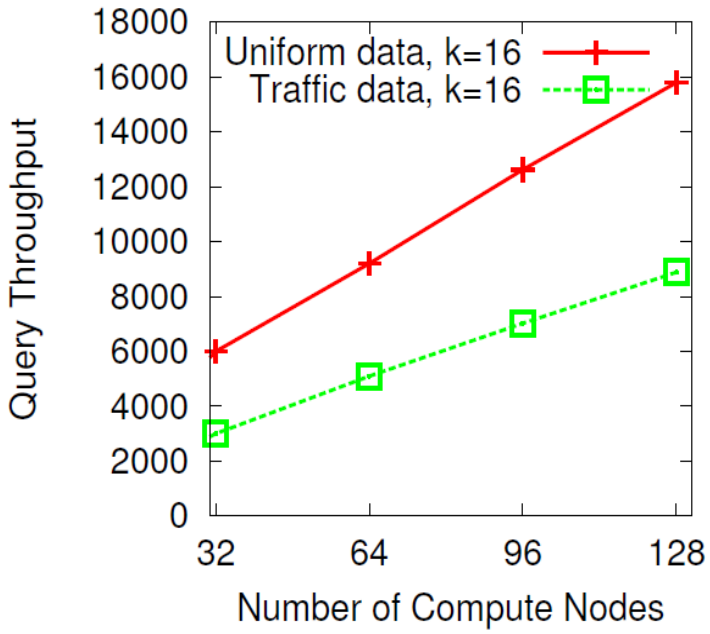

Figure 11. Performance of kNN Query (Traffic Dataset)

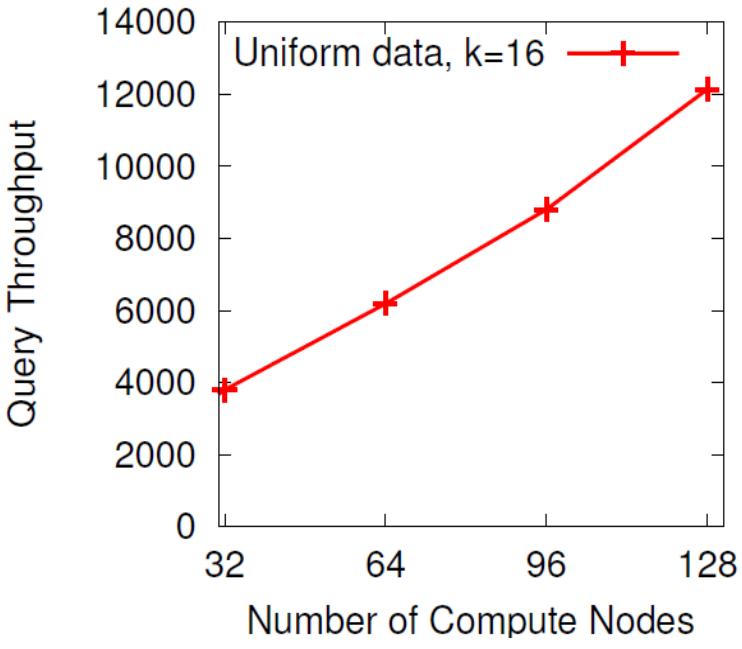

Figure 12. Performance of kNN Query (Uniform Dataset)

5. Conclusion

This paper designed cloud computing multidimensional indexing system (RTN). RTN is built on a local R tree, and in large scale to support efficient multidimensional data sharing cluster constructed in the cloud computing system to obtain. This paper used RTN to deal with a variety of multidimensional query algorithms, including range query and kNN query algorithm. Through the real cloud computing platform Amazon EC2 to test the performance of the RTN index, the experimental results show that the RTN index has good scalability. To effectively deal with all kinds of multidimensional queries in large-scale cloud, so RTN is a cloud computing system in an efficient multi-dimensional index structure.

References

[1] T. Feng, "Research on the reliability evaluation and task scheduling in cloud computing", University of Electronic Science and technology, **(2012)**.
[2] S. Chunju, "The cloud environment data model and index technology research", Nanjing University of Posts and Telecommunications, **(2013)**.
[3] Z. Peng, "The research of formal method in cloud computing", Jilin University, **(2014)**.
[4] D. Yuefa, "Study on data security technology of cloud computing based on multi dimension immune", The national defense science and Technology University, **(2009).**
[5] D. Yunyun, "Study on data index structure of multidimensional". Yunnan University, **(2014)**.
[6] C. Xinpeng, "Study on HBase data generation method based on index", Beijing University of Posts and Telecommunications, **(2014).**
[7] Y. Jianwei, "The study on cloud computing security technology operating system", Harbin Institute of Technology, **(2011)**.
[8] Z. Xiaoqin, "Study on model of network monitoring and trusted computing system" Chongqing University, **(2012)**
[9] W. Hongyu, "Research on the key technology of cloud computing in dynamic data migration", Dalian Maritime University, **(2010)**
[10] Y. Huan, "Study on the task scheduling of cloud computing environment", Henan University, **(2013)**.
[11] Z. Hua, "The cloud data storage model research and application", Hunan University, **(2013)**.
[12] X. Linhao, Z. Aoying, "Money Weining, unstructured P2P systems in multidimensional range search", Journal of software, **(2007)**,06:1443-1455.
[13] Z. Jianhui, "Realization of cloud computing oriented network caching system platform", Hangzhou Dianzi University, **(2013)**.

Authors

Qinghong Liu She received her B.S degree from North-east Normal University. She received her M.S degree from Jilin University. She is a professor in Jilin province economic management cadre college. Her research interests include Database, Computer application, Data mining.

Hongyan Zhang He received his B.S degree from Jilin Institute of Technology. He is an associate professor in Jilin province economic management cadre college. His research interests include Computer networks, intelligent algorithm optimization, Information Technology.

