International Journal of Automation and Computing 11(1), February 2014, 109-117
DOI: 10.1007/s11633-014-0772-y

A Multi-dimensional Index Structure Based on Improved VA-file and CAN in the Cloud

Chun-Ling Cheng[1,2,3] Chun-Ju Sun[1] Xiao-Long Xu[1,2] Deng-Yin Zhang[3]

[1]College of Computer, Nanjing University of Posts and Telecommunications, Nanjing 210003, China
[2]Jiangsu High Technology Research Key Laboratory for Wireless Sensor Networks, Nanjing 210003, China
[3]Key Lab of Broadband Wireless Communication and Sensor Network Technology, Nanjing University of Posts and Telecommunications, Ministry of Education Jiangsu Province, Nanjing 210003, China

Abstract: Currently, the cloud computing systems use simple key-value data processing, which cannot support similarity search effectively due to lack of efficient index structures, and with the increase of dimensionality, the existing tree-like index structures could lead to the problem of "the curse of dimensionality". In this paper, a novel VF-CAN indexing scheme is proposed. VF-CAN integrates content addressable network (CAN) based routing protocol and the improved vector approximation file (VA-file) index. There are two index levels in this scheme: global index and local index. The local index VAK-file is built for the data in each storage node. VAK-file is the k-means clustering result of VA-file approximation vectors according to their degree of proximity. Each cluster forms a separate local index file and each file stores the approximate vectors that are contained in the cluster. The vector of each cluster center is stored in the cluster center information file of corresponding storage node. In the global index, storage nodes are organized into an overlay network CAN, and in order to reduce the cost of calculation, only clustering information of local index is issued to the entire overlay network through the CAN interface. The experimental results show that VF-CAN reduces the index storage space and improves query performance effectively.

Keywords: Cloud computing, index, similarity search, clustering, vector approximation file (VA-file), content addressable network (CAN).

1 Introduction

With the rapid development of the computer and Internet technology, cloud computing, as a new computing platform, has emerged[1,2]. Though the unified definition of cloud computing has not been confirmed, it is considered as a revolution in IT industry and has attracted great interest from both academic and industrial communities. Due to the commercial potential of the cloud computing, many companies are increasing their investments in cloud research, such as Google′s cloud computing platform, Amazon elastic compute cloud (EC2)[3] and IBM′s blue cloud[4]. These cloud computing systems have a large number of computer nodes, store vast amounts of data, support large-scale data processing and retrieval applications, and provide extensible and reliable service to the end users.

In the cloud computing systems, data storage is mostly dependent on the underlying distributed file system (DFS)[5,6] to store data and adopts key-value-based[7−9] way to manage massive data. Dataset is divided into data chunks with equal size in DFS, and each data chunk is distributed in the computing node of cloud. Key-value model employs a simple key/value way to store data, where both key and value are arbitrary byte strings. This model does not analyse the specific content of data, which enables it to handle almost all types of data and support high scalability and mass data processing. However, it can only provide key-based insertion and query operations, which cannot effectively support complex queries. In the real world, users query operations are often not simple point query. For example, in YouTube, each video could be stored in a key-value store with a unique video ID as the key and video information, including title, upload time and number of views as the value[10]. Although the video can be efficiently retrieved via video ID, sometimes the end user wants to find videos with given titles or within a date range. Therefore, this storage model cannot support complex queries effectively, such as range query or similarity query, which is difficult to support personalized on-demand retrieval in cloud systems. Some tree-based indexing structures are proposed to support range query for data with different dimensions, however, they cannot support similarity search effectively and easily lead to the "curse of dimensionality" as the dimensionality increases.

This paper presents VF-CAN, an indexing structure for supporting similarity search. It combines content addressable network (CAN-based)[11] routing protocol and the improved vector approximation file (VA-file)[12] index. VF-CAN has two index levels: global index and local index. The data in cloud system are divided into data chunks and then stored on different computer nodes. To realize efficient local data management, each computer node builds its local index by an improved VA-file, named VAK-file. First the data are quantified and compressed into approximation vectors of VA-file, and then the approximation vectors are clustered by k-means. To avoid performance bottleneck, the

Manuscript received November 18, 2012; revised June 14, 2013.
This work was supported by National Natural Science Foundation of China (No. 61071093), Research and Innovation Projects for Graduates of Jiangsu Province (Nos. CXZZ12_0483 and CXLX12_0481), Science and Technology Support Program of Jiangsu Province (No. BE2012849), and Priority Academic Program Development of Jiangsu Higher Education Institutions (No. yx002001).

global index are distributed over each storage node which is organized into an overlay network CAN. Since the number of cluster centers has reduced significantly, it is feasible that only cluster information of local index is issued into the overlay network through CAN interface, which reduces the estimated cost of different index publishing strategies and improves query performance.

2 Related work

Indexing structure in the cloud environment can be divided into one-dimensional index and multi-dimensional index. For single-dimensional data, we can build one-dimensional index, such as B-tree index and hash index. A scalable distributed B-tree is proposed in order to support large scale data index in a cluster[13]. In this system, computer nodes can be divided into server nodes and client nodes. B-tree is distributed across servers, the client lazily replicates all inner B-tree nodes, and the servers synchronously maintain a B-tree version table for validation. The scalable distributed B-tree system uses distributed transactions to make changes to B-tree nodes, and B-tree nodes can be migrated online between servers for load-balancing. The main advantages of this index are massive scalability, low cost, fault-tolerance and manageability. But its weakness is that the client nodes need to lazily replicate all the corresponding internal nodes, which incurs high maintenance cost. A general indexing framework for cloud computing system is proposed in [14], and there are three layers in this framework. In the middle layer, computational resources are provided to users. Each node builds local index for the data stored in it to speed up the query, and establishes global index by selecting and publishing part of local indexes to overlay network. In the lower layer, processing nodes are organized in a structured overlay for routing and facilitating node′s joining and leaving. And in the upper layer, it provides a data access interface to the user′s applications based on the global index. The users can select different data access methods for different queries. Another hierarchical index structure is presented in [10] to process one-dimensional data in the cloud. First of all, it builds B-tree index for the local data stored in each node. Then, all the nodes are organized into a structured network BATON. In addition, it proposes an adaptive algorithm to select B+-tree node to publish according to query patterns. This index structure can support range query and improve query efficiency. Both of the index schemes in [10, 14] use two index levels, and build local index for the data stored in the slave nodes and global index in the server nodes. During querying, they locate the local index through the global index, and then search data in the slave nodes. Though these can reduce query time, they cannot support multi-dimensional query effectively. Unlike tree index structure, bitmap index is affected slightly by data growth, therefore, a bitmap-based indexing mechanism is proposed in [15]. It is built for the local data, with BATON to organize its nodes. The advantage of this scheme is that bitmap indexes can improve query efficiency, however, it is better than the B-tree index only when there are a lot of duplicate values in a column and when to use "and" and "or" operations. Reference [16] proposed a regional bitmap index, which is a secondary index for data management in cloud computing environment. This index structure combines the advantages of centralized and distributed logical secondary indexing structures, and it has good scalability.

Several indexing structures for multi-dimensional data are proposed in [17−21], which use two index levels. efficient multi-dimensional index with node cube (EMINC)[17] uses the combination of R-tree and KD-tree to organize data and develops the node bounding technique to reduce query processing cost on the cloud platform. Moreover, an index update strategy based on cost estimation is also proposed to reduce the cost of index publishing. However, in the relative nodes locating phase, EMINC chooses all the slave nodes in the cluster as the candidates of the query, which leads to high cost. RT-CAN[18] introduces an efficient multi-dimensional index structure to support complex queries in a cloud system based on the routing protocol of CAN and R-tree indexing scheme. RT-CAN uses R-tree to build index for the data that are locally stored and CAN to organize storage nodes. Furthermore, it proposes a query-conscious cost model to select beneficial local R-tree nodes for publishing. But the query performance of R-tree will decrease significantly for high-dimensional data management. In addition, an extensible framework for implementing database management systems (DBMS)-like indexes in the cloud is proposed in [19] and it defines a methodology to map various types of data and peer to peer (P2P) overlays to a generalized Cayley graph structure which can support a variety of index structure flexibly, such as hash index and B-tree index. Finally, an adaptive strategy to optimize index performance is proposed. A novel quad-tree based multi-dimensional index structure is proposed in [20] for efficient data management and query processing in cloud computing systems. It also adopts two index levels. Each computer node builds local quad-tree index to manage the data residing in it. And all computer nodes are organized in a chord-based overlay network. Reference [21] introduces a complemental clustering index: CCIndex, which creates several complemental clustering index tables for performance, leverages region-to-server information to estimate result size, and supports incremental data recovery. CCIndex can support multi-dimensional range queries and solve the issues of high performance, low space overhead, and high reliability.

The indexes in cloud environment mentioned above are mainly a tree-based structure that will incur "the curse of dimensionality" problem with the increases in dimensionality. In traditional distributed computing, Weber et al.[12] proposed VA-file index to improve query efficiency by quantifying and compressing the high-dimensional data and implemented it. The storage space of VA-file is much smaller than that of the original high-dimensional data. VA-file has two significant advantages: first, the size of index file is much smaller than the original file, so disk input/output (I/O) cost is greatly reduced during a sequential scan; second, it reduces the computational complexity. However it must scan the entire approximation vector during querying, thus the cost becomes relatively large when data volume is huge. VAR-tree is presented in [22] to solve "the curse of dimensionality" problem. It integrates VA-file and R-tree, and adopts R-tree to manage the approximations. A k-nearest neighbor (KNN) search algorithm is also proposed to perform similarity in a VAR-tree, which improves retrieval performance. Inspired by [22], we present a two level index structure by improving VA-file and integrating CAN to support similarity search.

3 VF-CAN index

The index structure of VF-CAN has two levels: global index level and local index level. Local index is VAK-file, which is built for the data stored in each storage node. VAK-file index is the improvement of VA-file index by using k-means to cluster VA-file vectors according to their degree of proximity. The global index is built on top of the local index and it includes cluster information of the local index. Each storage node issues its local cluster center information and IP address to the overlay network through the interface of CAN. The key problem of the VF-CAN index lies in how to build local index and issue local index into global index.

3.1 System architecture

In our cloud environment, dataset is divided into many data chunks according to horizontal partitioning scheme. These data chunks are distributed over different servers in the data centers, and each server is both a storage node and an overlay network node. The storage node is a node in distributed storage system of the cloud data center for storing original data chunks, local index and part of the global index. In order to avoid the problem of performance bottlenecks, the global index is distributed in storage nodes. Meanwhile, in order to facilitate management, each storage node is organized with a structure overlay CAN to manage global index. Thus the storage node is also an overlay network node logically corresponding to a partition of CAN. Fig. 1 shows the architecture of the system.

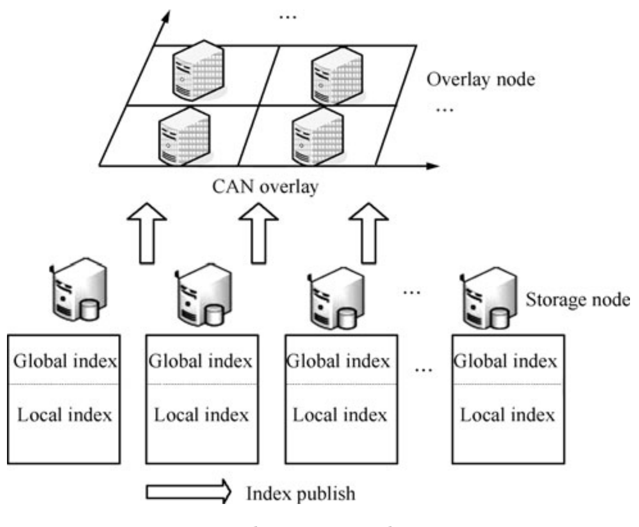


Fig. 1 The system architecture

In Fig. 1, in order to facilitate search, each storage node builds VAK-file index for its local data and shares a portion of its storage space to maintain global index. The cluster center information from local index is published into the overlay network via the interface of CAN to form the global index. The format of published information is (ip, u_j), where ip is IP address of the corresponding storage node and u_j is the corresponding cluster center of j-th cluster in local index.

Similarity query processing can be divided into two phases. In the first phase, the global index is looked up by CAN routing mechanism and the entries that satisfy the query are returned. In the second phase, based on the received index entries, the query is forwarded to the storage node via the IP address, on which the corresponding cluster u_j in local index is found out through cluster center information file. At last, the top-K data items which are closest tou_j in the cluster file would be returned.

3.2 Local index VAK-file

As multimedia data become the vector data after abstraction, "similarity search" has converted into similar vectors query of high-dimensional space. VA-file improves query performance by quantifying and compressing the high-dimensional data, however it must scan the entire approximation vectors during querying, which will result in high cost when data volume is large. In this paper, VA-file approximation vectors are clustered by k-means according to their degree of proximity. Each cluster forms a separate index file which stores the approximate vectors contained in the cluster, and the cluster center is stored in the cluster center information file of the corresponding storage node. In this way, the query of the second phase will only be forwarded to the corresponding cluster without scanning the entire approximate vectors, which improves the query efficiency.

Assuming that the dimensionality of initial vector data which are abstracted from multimedia data is n, and there are m records in all. The algorithm to establish the local index VAK-file is described as follows:

Algorithm 1. The establishment of local index VAK-file

Step 1. Quantify and compress the initial vector data, as VA-file, assuming that the approximate vector set after compression is $V = \{V_1, \cdots, V_m\}$.

Step 2. Cluster the approximate vector set V.

1) Select k approximation vectors randomly from m approximation vectors as initial cluster centers, namely: $u_1, u_2, \cdots, u_k \in V$.

2) For the rest of the m-k approximation vectors, calculate their distance from the cluster centers by the Euclidean distance:

$$d_{ij} = \mathrm{sqrt}(\sum_{r=1}^{n}(V_{ir} - u_{jr})^2) \tag{1}$$

where $1 \leqslant i \leqslant m-k$, $1 \leqslant j \leqslant k$, $1 \leqslant r \leqslant n$. d_{ij} indicates the distance between the i-th approximation vector and the j-th cluster center vector. V_{ir} denotes the r-th dimensional data of the i-th approximation vector, and u_{jr} denotes the

r-th dimensional data of the j-th approximation vector.

3) Assign each vector to its nearest cluster; for each approximate vector, calculate the cluster which should belong to

$$c_i = \arg\max_j (d_{ij}) \tag{2}$$

here, $1 \leqslant j \leqslant k$. c_i represents the cluster that is nearest to the approximate vector V_i.

4) Update each cluster center u_j:

$$u_j = \frac{1}{N_j} \sum_{i=1}^{N_j} V_{ij} \tag{3}$$

where $1 \leqslant i \leqslant N_j$, N_j is the number of approximate vectors in the j-th cluster, u_j indicates the j-th cluster center, V_{ij} is the i-th approximate vector of the j-th cluster.

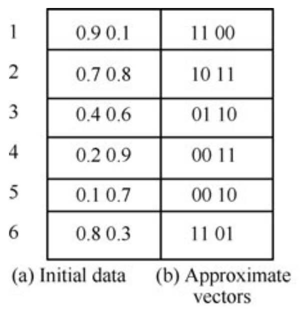
5) Repeat steps 2) – 3) until the standard measure function begins to converge. Here we use mean-square deviation as standard measure function, as follows:

$$\sigma_j = \mathrm{sqrt}\left(\frac{1}{N_j} \sum_{i=1}^{N_j} (V_i - u_j)^2\right) \tag{4}$$

where σ_j indicates the mean-square deviation of the j-th cluster. N_j is the number of approximate vectors belonging to the j-th cluster.

Step 3. The cluster center and the number of each cluster are stored in the cluster center information file.

The following is an example of local index. Fig. 2 (a) shows the initial vector data which are abstracted from multimedia data. Dimensionality n is 2 and the number of records is 6.



	(a) Initial data	(b) Approximate vectors
1	0.9 0.1	11 00
2	0.7 0.8	10 11
3	0.4 0.6	01 10
4	0.2 0.9	00 11
5	0.1 0.7	00 10
6	0.8 0.3	11 01

Fig. 2 Example of approximations

Fig. 2 (b) shows the approximate vectors after compression according to Step 1 of algorithm when the number of clusters is $k = 2$.

Fig. 3 shows the clustering results of the approximate vectors in Fig. 2 according to Step 2.

In Fig. 3 (a), initial data are clustered into two classes. Fig. 3 (b) shows two cluster files u_1 and u_2 that store the corresponding approximate vectors, and each cluster corresponds to a cluster center. The cluster center information file is displayed in Table 1.

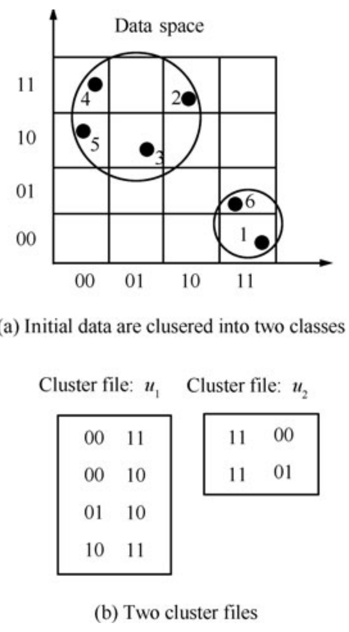


Fig. 3 Example of approximations clustering

Table 1 Example of cluster center information file

Cluster center	Cluster number
⟨0.75, 2.5⟩	u_1
⟨3.0, 0.5⟩	u_2

There are two clusters in Table 1 which include the center and number of each cluster.

3.3 Global index

1) Content addressable network (CAN)

VF-CAN organizes storage nodes into an overlay network CAN for better routing, because none of the other overlays except CAN can support multi-dimensional query naturally. CAN[11] is a scalable, self-organized structured peer-to-peer overlay network. A d-dimensional CAN partitions a virtual d-dimensional Cartesian coordinate space containing all nodes, and assigns each node a d-dimensional zone. A node in CAN maintains data mapped to its zone. Key of each data is mapped to a point P in the coordinate space based on distributed hash table (DHT). Data item (*key*, *value*) is stored in the CAN node whose zone contains the point P. Each node in CAN maintains a routing table which contains the IP address that is adjacent to this node. According to this routing table, user′s query can be routed between any two points of the coordinate space. Once node gets a query for a certain *key*, first it maps the data item via its key to a point P in virtual coordinate space based on DHT, the query destination address is the coordinate of P; and then it routes the query to a neighbor whose zone is nearest to the point P, the query result will be returned by the neighbor. In a d-dimensional CAN with N nodes, the average number of routing hops for a query is $\frac{d}{4}N^{\frac{1}{d}}$, and each node maintains $2d$ neighbors in its routing table[18].

2) Publishing in the VF-CAN index

The size of local index is proportional to the amount of data stored. We cannot issue all the approximate vectors into global index in order to reduce maintenance cost. In our work, each cluster corresponds to a cluster center, and the number of cluster centers is significantly less than the total number of approximate vectors, thus only the cluster center information of the local index is issued. The process is as follows:

Each storage node, in accordance with the CAN node mapping algorithm, maps the clustering center u_j, the key of $(ip,\ u_j)$, to a point P in the coordinate space based on DHT, that is $P = hash_{\mathrm{CAN}}(u_j)$, then $(ip,\ u_j)$ is stored in the CAN node whose zone contains the point P. Once node gets a query for a certain u_j, it can use the same hash function to find corresponding point P according to u_j, and then get the relevant value which corresponds to u_j from the node whose zone contains P. All the clustering information of all storage nodes is issued, so we can locate every vector of local index according to the global index.

For example, in Fig. 3 (b) and Table 1:

The first cluster center is $\langle 0.75,\ 2.5\rangle$, and the second is $\langle 3.0,\ 0.5\rangle$, we issue $(ip1,\ \langle 0.75,\ 2.5\rangle)$ and $(ip1,\ \langle 3.0,\ 0.5\rangle)$ to global index, where $ip1$ is the IP address of corresponding storage node. Suppose the 2-dimensional coordinate space is divided into nine areas, as shown in Fig. 4, virtual coordinate space of each partition is dynamically allocated to each node of CAN. The index publishing strategy above is that the clustering center information of local index is mapped to a node whose zone includes the center area. For example, the clustering center of $(ip1,\ \langle 0.75,\ 2.5\rangle)$ is $\langle 0.75,\ 2.5\rangle$, through $P = hash_{\mathrm{CAN}}(\langle 0.75, 2.5\rangle)$, it would get point P which corresponds to node A in Fig. 4. So $(ip1,\ \langle 0.75,\ 2.5\rangle)$ would be published to node A whose zone contains the area of $\langle 0.75,\ 2.5\rangle$. In the same way, $(ip1,\ \langle 3.0,\ 0.5\rangle)$ would be issued to the node B.

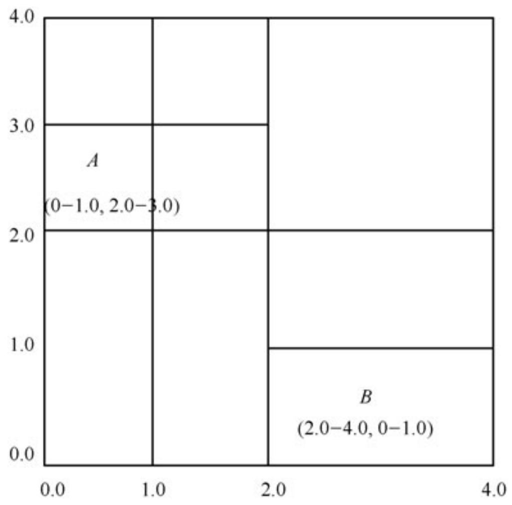


Fig. 4 2-dimensional virtual coordinate

3.4 Comparison of several indexes

In this section, we will compare VF-CAN against several typical indexes. Existing index structures for cloud environment are mostly based on the B-tree, KD-tree and R-tree.

The B-tree is a balanced m-way search tree which is an extension of a binary search tree. A B-tree of order m has the following characteristics: 1) Each node has m children at most. 2) The root node has at least two children if it is not a leaf node. 3) Each non-leaf node (except root) has $\lceil m/2 \rceil$ children at least. 4) Each path from the root to the leaf has the same length which means that all leaf nodes are at the same level. The main advantage of B-tree is that it is fast and simple; however it cannot handle multi-dimensional data.

KD-tree is the advancement of the binary tree in multi-dimensional data space, and it is the K-dimensional binary tree. All the non-leaf nodes can be regarded as implicitly generating a hyperplane that divides the space into two parts, known as half-spaces. Points on the left of this hyperplane represent the left subtree of that node, and points on the right of the hyperplane represent the right subtree. But as an unbalanced tree, with the increase of the amount of data, the height of KD-tree increases rapidly which is unsuitable for the query of vast amounts of data.

R-tree family is a kind of balanced tree with a hierarchical structure similar to B-tree, which mainly includes R+-tree and R*-tree. The data stored in the R-tree are the minimum bounding rectangle (MBR) of these data, instead of the original data. The shortcoming of R-tree is that it is prone to cause the "curse of dimensionality" in processing high-dimensional data and it cannot guarantee the success with one path search during lookup operation.

VF-CAN in this paper first quantifies and compresses the high-dimensional data with bit strings according to the method of VA-file. Each dimension of data space is quantified into a number of intervals which are separated by grid lines and indicated by bit strings with a small space. The corresponding bit strings of each dimension are connected as approximation vectors to represent high-dimensional data. After that, approximation vectors are clustered by k-means according to their similarity degree. The quantifying and compressing of high-dimensional data reduce the data storage space significantly, and the clustering of approximation vectors improves query performance.

4 KNN query in VF-CAN

The range query and KNN query of multi-dimensional data are collectively referred to as "similar" query[22], however, the KNN query is more popular in the majority of practical applications. Given a KNN query $Query(key, K)$, our algorithm will return the top-K data items which are closest to *key*. In order to distinguish the same parameters k between k-means algorithm and KNN query, we use lowercase k in k-means and uppercase K in KNN.

As discussed before, we locate the local index according to global index, and then the query is routed to the corresponding storage node and processed locally, which can reduce search time significantly.

The algorithm is as follows:

Algorithm 2. KNN query processing in VF-CAN

Step 1. Calculate the approximate vector V' of vector data key for the query.

Step 2. Query the global index according to CAN routing mechanism, calculate the distance d between approximate vector V' and each cluster center, and return cluster $C_i\,(ip, u_j)$ of the minimum distance.

Step 3. Locate the corresponding storage node and cluster according to the IP address and u_j of $C_i\,(ip, u_j)$, and then do the KNN query in cluster u_j.

Step 4. If the number of the returned data items is less than K, then jump to Step 2 and select the suboptimal cluster and continue to query. Otherwise, the query completes.

5 Experiments and results analysis

The establishment of index is to accelerate the speed of data query, so query efficiency is an important indicator to reflect index structure performance. In this section, we evaluate the performance of our scheme from four aspects: the size of index structure file, query response time, query throughput and the quality of query results. The size of index structure file reflects the space cost. Query response time and throughput are used to evaluate the speed and scalability of this scheme. The quality of query results reflects the similarity between query results and the query data. In the following experiments, we will compare the performance of our VF-CAN with that of the original VA-file index and RT-CAN, a typical index of the tree structure existing in cloud environment.

Our testing infrastructure includes 3 machines which are connected together to simulate cloud computing platforms in laboratory LAN. One machine plays the role of master node which is responsible for the start of the program and task allocation, the other two simulate 32 to 128 slave nodes which are responsible for program execution and the results will be returned to master node. Communication bandwidth is 10 Mbps, each machine has a 2.40 Hz Inter(R) Core(TM) i5 CPU, 2.00 GB of memory, and a 320 G disk. We use Java 1.0.0.17 to implement our scheme. The test data is high dimensional feature database ColorHistogram.asc[23] extracted from the Corel image set. It is a general dataset for multi-dimensional data management. The dimensionality of the dataset is 32 with a total of 68 040 records. It can be downloaded from http://kdd.ics.uci.edu.

5.1 Size of index file

In order to compare the size of the index file, we extract 10 000 rows from the test data to form data file Vector1.dat, and extract 68 040 rows to form file Vector2.dat. Table 2 shows the data file after extraction and the size information of corresponding local index.

Table 2 File size corresponding to different index structures (MB)

Index file	Vector1.dat	Vector2.dat
Original vector file	2.94	20.0
VA-file	0.44	3.02
VF-CAN	0.48	3.11
RT-CAN	4.74	32.2

Table 2 shows that the space occupied by the VF-CAN is slightly larger than the VA-file, and this is because that VF-CAN needs to store the cluster center information. But the space occupied by the VF-CAN is much smaller than RT-CAN index structure and the original file. Therefore, it is easier to store it into memory to improve query efficiency.

5.2 Query response time

Response time is the time from submitting a query request to the results returned. It reflects the influence of the index structure on the query speed, and is an important indicator to evaluate index structure performance. Therefore, in order to verify the response time of VF-CAN, we randomly select the 2-dimensional, 5-dimensional, 20-dimensional, 32-dimensional data from the original database, and select 100 records as query samples. Here, the number of nodes is set to 32, K in KNN is 50. The average response time of 100 times query tests is shown in Fig. 5.

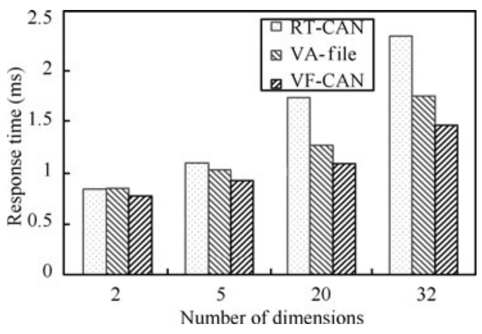


Fig. 5 Response time with different dimensionalities

It can be seen in Fig. 5 that when data dimension is small, the difference in three indexes′ response time is not obvious. With the increasing of the dimensionality, query response time of three different index structures all increases, but the advantage of the VF-CAN is becoming more obvious, which is due to the fact that VF-CAN does not need to scan all of the vector data, and only query in the corresponding cluster, therefore, its response time is shorter than that of the VA-file and RT-CAN. In RT-CAN structure, with the increasing of the dimensionality, the boundary rectangle overlap of R-tree increases dramatically, which leads to a rapid decrease in performance and larger increase in response time.

5.3 Query throughput

Query throughput represents the number of queries processed per second, which reflects the number of queries that index structure can support. It is also an important factor to evaluate the performance of index structure. In this section, we compare the throughput of the KNN queries of three index structures when the amount of data, value of K in KNN, data dimensionality and the number of nodes are different respectively. 100 records are randomly selected from the database as query samples, and we use the average throughput of 100 times query tests on the basis of their performance.

First of all, in order to evaluate the performance of VF-CAN in different data volume, we randomly select 500, 2 000, 10 000, 30 000, 50 000 data records from the 32-dimensional original feature database, and compare the throughput of three index structures when data quantity gradually becomes larger. The experimental results are shown in Fig. 6, where the number of nodes is 32, K in KNN is 50.

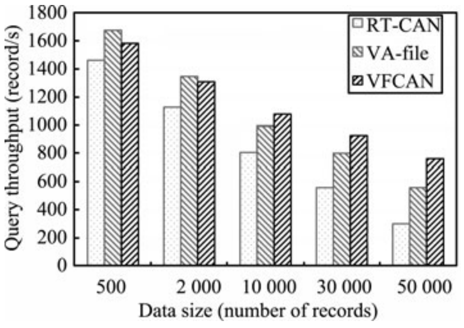


Fig. 6 Corresponding throughput of different data size

As can be seen in Fig. 6, with the increase of data volume, the throughput of three different index structures all decreases. Due to the overlap of bounding rectangle in R-tree under high-dimensionality of data, RT-CAN′s performance is the worst. When the amount of data is small, we find that VA-file performs better than VF-CAN, which is because the cost of scanning all data in VA-file is less than that of VF-CAN which should cluster the vector data. However, as the amount of data increases, the advantage of VF-CAN is more and more obvious. It only queries in corresponding cluster without scanning all of the vector data, thus the throughput is higher than that of the VA-file.

Secondly, in KNN query, the value of K has an impact on query performance. So we select the 2-dimensional and 32-dimensional data from the original database respectively, and compare the query throughput of three index structures when K is 1, 10, 20, 30, 40, and 50, respectively. The experimental results are shown in Fig. 7, where the node number is set to 32.

We can see in Figs. 7 (a) and (b) that with the increase of K, the throughput of three index structures all gradually reduces. This is because with the increase of K, query space also increases and the amount of data needed to query is larger. In Fig. 7 (a) corresponding to 32-dimensional data, RT-CAN uses tree-based index structure, and will cause "curse of dimensionality" in the high dimensional case, so its performance is worse than that of VA-file and VF-CAN. VF-CAN, which adopts clustering and two levels index, does not need to scan all the vectors and support more queries in unit time, therefore, its performance is better than that of the VA-file. Fig. 7 (b) shows that the throughput of 2-dimensional data is better than that of 32-dimensional data, but the trend of throughput in Figs. 7 (a) and (b) are the same.

Then, we compare the throughput of three index structures with different number of nodes. When test data is the 32-dimensional feature database, the nodes number is 32, 64, 96, and 128, respectively. The experimental results are shown in Fig. 8 below.

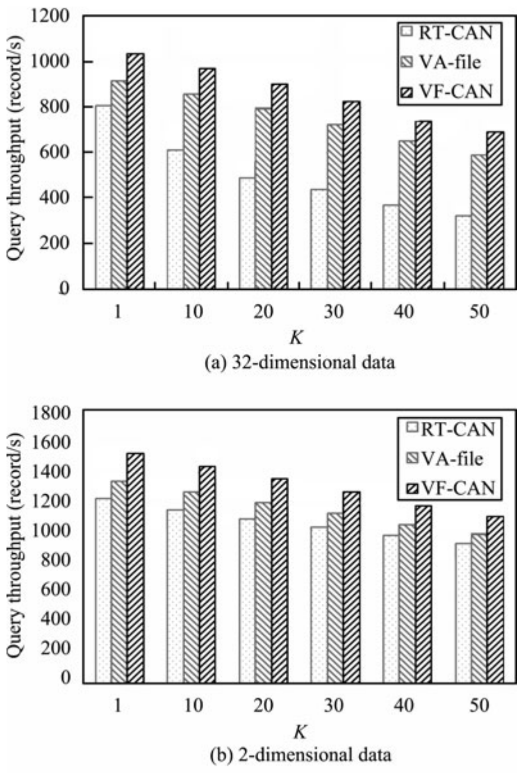


Fig. 7 Query throughput with different K

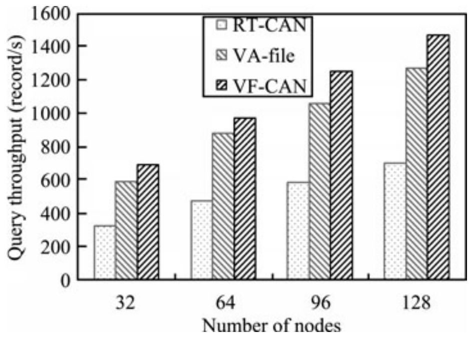


Fig. 8 Corresponding throughput of different node number

We can see in Fig. 8 that the query throughput of three different index structures will all increase when the number of nodes increases. VA-file uses a quantizing and compressing method, therefore, its performance is significantly better than that of RT-CAN. Due to the frequent splitting of tree node, RT-CAN has low performance. As VF-CAN clusters approximate vector data, it does not need to scan all data, therefore, its throughput is better than that of the VA-file.

At last, in order to compare the query throughput of various numbers of dimensions, we randomly select the 2-dimensional, 5-dimensional, 20-dimensional, 32-dimensional data from the original database for experiment. Here, the node number is 32. The experimental results are shown in Fig. 9.

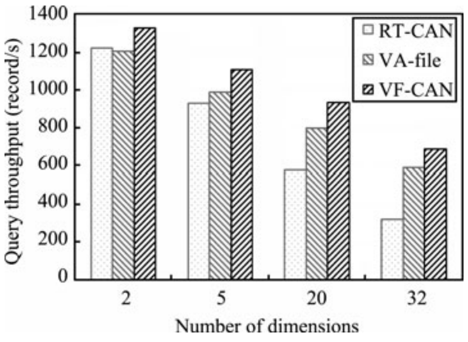


Fig. 9 Query throughput with different dimensionalities

It can be seen in Fig. 9 that when the dimensionality is 2, the performance of the RT-CAN and the VA-file are almost the same. However as the dimensionality increases, R-tree′s performance decreases rapidly because the overlap of its boundary rectangle increases sharply, and the cost of index publishing strategy becomes larger. Therefore the performance of VA-file and VF-CAN are significantly higher than that of the tree-based structure RT-CAN.

5.4 Quality of query results in VF-CAN

For similarity search, the quality of query results is an important factor in evaluation of the index structure and query algorithm, and the general evaluation criteria are ε and ρ[22]. ε denotes the difference between the query result set and the actual data. The smaller ε is, the higher similarity is. ρ represents accuracy of the result set, and a bigger ρ means that there are more accurate data appearing in the result set. ε and ρ are calculated as following[22]:

$$\varepsilon = \frac{\sum\limits_{p\in R_{\rm AKNN}} D(p,q) - \sum\limits_{p\in R_{\rm KNN}} D(p,q)}{\sum\limits_{p\in R_{\rm KNN}} D(p,q)} \tag{5}$$

$$\rho = \frac{||R_{\rm AKNN} \cap R_{\rm KNN}||}{||R_{\rm KNN}||} \tag{6}$$

where $R_{\rm KNN}$ means accurate KNN query, R_{AKNN} is approximate KNN query, q is the query vector and p is the vector in the testing dataset. We randomly select 2-dimensional, 5-dimensional, 20-dimensional, and 32-dimensional data from our dataset that we have downloaded as testing dataset, and 100 records as query samples. We conduct a similarity query and precise query experiment when K is 50, and calculate the average of ε and the average of ρ. Table 3 shows the experiment results.

Table 3 Quality of ε and ρ

Number of	K=10		K=20		K=50	
dimensions	ε	ρ(%)	ε	ρ(%)	ε	ρ(%)
2	0.0196	80.1	0.0187	81.7	0.0193	81.2
5	0.0184	81.3	0.0179	83.8	0.0181	83.4
20	0.0173	83.2	0.0166	85.3	0.0164	87.6
32	0.0152	85.7	0.0144	87.5	0.0105	90.6

In Table 3, we can see that, under different circumstances, ε is very small which means that the difference is very small, therefore the data are very similar. Meanwhile, the precision of ρ is more than 80 %, which indicates that vast majority data of accurate result set are concentrated in approximate results. This result can satisfy the requirement of most applications for the "similar" query.

The experimental results show that the storage space of proposed VF-CAN index is slightly larger than that of the VA-file, however it is much smaller than that of R-tree and the original file. Moreover, the throughput of the proposed system in different situations has relatively increased and response time has decreased, which improves data query performance effectively.

6 Conclusions

In this paper, we present VF-CAN, a novel multi-dimensional indexing scheme, for supporting similarity search in cloud system. Considering the characteristics of data storage in cloud environment, VF-CAN has two index levels: global index and local index. In the local index VAK-file, VA-file approximation vectors are clustered by k-means according to their degree of proximity, which effectively solves the problem that incurs high cost when data volume is large. Each storage node issues its cluster center information and IP address of the local index as global index in order to reduce load and improve performance. Meanwhile, storage nodes are organized into a structured overlay network CAN for better routing. The experimental results show that VF-CAN can improve query performance effectively. This paper focuses on the establishment of the local index and publishing of global index, and concentrates less on the index maintenance. So the future work is to improve the index maintenance efficiency.

References

[1] Y. C. Liu, Y. T. Ma, H. S. Zhang, D. Y. Li, G. S. Chen. A method for trust management in cloud computing: Data coloring by cloud watermarking. *International Journal of Automation and Computing*, vol. 8, no. 3, pp. 280–285, 2011.

[2] Y. K. Guo, L. Guo. IC cloud: Enabling compositional cloud. *International Journal of Automation and Computing*, vol. 8, no. 3, pp. 269–279, 2011.

[3] M. Lynch. Amazon Elastic Compute Cloud (Amazon EC2), [Online], Available: http://aws.amazon.com/ec2/.

[4] K. Chen, W. M. Zheng. Cloud computing: System instances and current research. *Journal of Software*, vol. 20, no. 5, pp. 1337–1348, 2009.

[5] S. Ghemawat, H. Gobioff, S. T. Leung. The Google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, ACM, New York, USA, pp. 29–43, 2003.

[6] K. Shvachko, H. R. Kuang, S. Radia, R. Chansler. The hadoop distributed file system. In *Proceedings of the 26th IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, IEEE, Lake Tahoe, NV, USA, pp. 1–10, 2010.

[7] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels. Dynamo: Amazon′s highly available key-value store. In *Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles*, ACM, New York, USA, pp. 205–220, 2007.

[8] Project Voldemort: Voldemort is a Distributed Key-value Storage System, [Online], Available: http://project-voldemort.com/, February 18, 2013.

[9] Apache Cassandra, [Online], Available: http://cassandra.apache.org/, February 18, 2013.

[10] S. Wu, D. W. Jiang, B. C. Ooi, K. L. Wu. Efficient B-tree based indexing for cloud data processing. *Proceedings of the VLDB Endowment*, vol. 3, no. 1–2, pp. 1207–1218, 2010.

[11] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker. A scalable content-addressable network. In *Proceedings of 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ACM, New York, USA, pp. 161–172, 2001.

[12] R. Weber, H. J. Schek, S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24th International Conference on Very Large Data Bases*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 194–205, 1998.

[13] M. K. Aguilera, W. Golab, M. A. Shah. A practical scalable distributed B-tree. *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 598–609, 2008.

[14] S. Wu, K. L. Wu. An indexing framework for efficient retrieval on the cloud. *IEEE Data Engineering Bulletin*, vol. 32, no. 1, pp. 75–82, 2009.

[15] B. Huang, Y. X. Peng. An efficient two-level bitmap index for cloud data management. In *Proceedings of the 3rd IEEE International Conference on Communication Software and Networks (ICCSN)*, IEEE, Xi′an, China, pp. 509–513, 2011.

[16] B. P. Meng, T. J. Wang, H. Y. Li, D. Q. Yang. Regional bitmap index: A secondary index for data management in cloud computing environment. *Chinese Journal of Computers*, vol. 35, no. 11, pp. 2306–2316, 2012. (in Chinese)

[17] X. Y. Zhang, J. Ai, Z. Y. Wang, J. H. Lu, X. F. Meng. An efficient multi-dimensional index for cloud data management. In *Proceedings of the 1st International Workshop on Cloud Data Management*, ACM, New York, USA, pp. 17–24, 2009.

[18] J. B. Wang, S. Wu, H. Gao, J. Z. Li, B. C. Ooi. Indexing multi-dimensional data in a cloud system. In *Proceedings of 2010 ACM SIGMOD International Conference on Management of Data*, ACM, New York, USA, pp. 591–602, 2010.

[19] G. Chen, H. T. Vo, S. Wu, B. C. Ooi, M. T. Özsu. A framework for supporting DBMS-like indexes in the cloud. *Proceedings of the VLDB Endowment*, vol. 4, no. 11, pp. 702–713, 2011.

[20] L. L. Ding, B. Y. Qiao, G. R. Wang, C. Chen. An efficient quad-tree based index structure for cloud data management. *Web-Age Information Management*, Berlin, Heidelberg: Springer, pp. 238–250, 2011.

[21] Y. Q. Zou, J. Liu, S. C. Wang, L. Zha, Z. W. Xu. CCIndex: A complemental clustering index on distributed ordered tables for multi-dimensional range queries. *Network and Parallel Computing*, Berlin, Heidelberg: Springer, pp. 247–261, 2010.

[22] D. G. Dong. High-dimensional Data Index Structure Research, Ph. D. dissertation, Fudan University, China, 2005. (in Chinese)

[23] Irvine. UCI Knowledge Discovery in Databases Archive, [Online], Available: http://kdd.ics.uci.edu, February 18, 2013.

Chun-Ling Cheng received the B. Sc. degree in computer software and M. Sc. degree in computer application from Nanjing University of Science and Technology, Jiangsu, China in 1993 and 1996, respectively. She is currently a Ph. D. candidate and an associate professor at the College of Computer, Nanjing University of Posts and Telecommunications, China.

Her research interests include data management, cloud computing and network management.

E-mail: chengcl@njupt.edu.cn (Corresponding author)

Chun-Ju Sun received the B. Sc. degree in computer application from Yancheng Institute of Technology, China in 2010. She is currently working toward the M. Sc. degree in computer application at Nanjing University of Posts and Telecommunications, China.

Her research interests include data management in cloud computing.

E-mail: sun.0806@163.com

Xiao-Long Xu received the M. Sc. and Ph. D. degrees in computer science and technology from Nanjing University of Posts and Telecommunications, China in 2002 and 2008, respectively. He is currently an associate professor at the College of Computer, Nanjing University of Posts and Telecommunications.

His research interests include computer software, distributed computing, and information security.

E-mail: xuxl@njupt.edu.cn

Deng-Yin Zhang received the B. Sc. degree in telecommunication engineering, the M. Sc. degree in circuit, signal and system, and the Ph. D. degree in signal and information processing from Nanjing University of Posts and Telecommunications, China in 1986, 1989, and 2004, respectively. He is currently a Ph. D. supervisor and researcher of the Key Lab of Broadband Wireless Communication and Sensor Network Technology.

His research interests include information network and information processing.

E-mail: zhangdy@njupt.edu.cn