Efficient R-Tree Based Indexing for Cloud Storage System with Dual-Port Servers*

Fan Li, Wanchao Liang, Xiaofeng Gao**, Bin Yao, and Guihai Chen

Department of Computer Science and Engineering,
Shanghai Jiao Tong University, P.R. China

Abstract. Cloud storage system such as Amazon's Dynamo and Google's GFS poses new challenges to the community to support efficient query processing for various applications. In this paper we propose RT-HCN, a distributed indexing scheme for multi-dimensional query processing in *data centers*, the infrastructure to build cloud systems. RT-HCN is a two-layer indexing scheme, which integrates HCN-based routing protocol and the R-Tree based indexing technology, and is portionably distributed on every server. Based on the characteristics of HCN, we design a special index publishing rule and query processing algorithms to guarantee efficient data management for the whole network. We prove theoretically that RT-HCN is both query-efficient and space-efficient, by which each server will only maintain a tolerable number of indices while a large number of users can concurrently process queries with low routing cost. We compare our design with RT-CAN, a similar design in traditional P2P network. Experiments validate the efficiency of our proposed scheme and depict its potential implementation in data centers.

Keywords: Distributed Index, R-Tree, Data Center Network.

1 Introduction

Cloud storage systems have been continuously drawing attentions from both academia and industry in recent years. From classical systems for general data services, such as Google's GFS [1], Amazon's Dynamo [2], Facebook's Cassandra [3], to newly designed systems with specialities, such as Haystack [4], Megastore [5], Spanner [6], various distributed storage systems were constructed to satisfy the increasing demand of online data-intensive applications that require massive scalability, efficient manageability, reliable availability, and low latency in the storage layer. Correspondingly, many works are proposed for designing

* This work has been supported in part by the National Natural Science Foundation of China (Grant number 61202024, 61202025, 61133006), China 973 project (2014CB340303, 2012CB316200), Shanghai Educational Development Foundation (Chenguang Grant No.12CG09), Shanghai Pujiang Program 13PJ1403900, and the Natural Science Foundation of Shanghai (Grant No.12ZR1445000).

** Corresponding author.

H. Decker et al. (Eds.): DEXA 2014, Part II, LNCS 8645, pp. 375–391, 2014.

new indexing scheme and data management system to support large-scale data analytical jobs and high concurrent OLTP queries [7–10].

In a cloud DB system, datasets are partitioned among distributed servers, while users may hop among multiple servers to process their query requests. To provide an efficient indexing scheme, a common feature of the above literature is that they split indices into two categories: global index and local index, and then portioned their global indices to each server according to an overlay network architecture. Following the direction of indices, users route queries among servers based on the underlying routing protocols of the network connecting servers together. However, all these designs are constructed on P2P networks [10–12], while nowadays cloud systems are usually built on an infrastructure called *data center*, which consists of large number of servers interconnected by a specific *Data Center Network* (DCN) [13–20]. For instance, Cisco applies Fat-Tree topology as its DCN architecture for provably efficient communication [13]. Different from P2P network, DCN is more structured with low equipment cost, high network capacity, and support of incremental expansion. It is natural that such infrastructures bring new challenges for researchers to design efficient indexing scheme to support query processing for various applications.

In this paper, we propose our RT-HCN, a distributed indexing scheme for multi-dimensional query processing in *Hierarchical Irregular Compound Networks* (HCN) [21, 22], which is the latest designed DCN structure using dual-port servers. HCN has many attractive features including low diameter, high bisection width, large number of node-disjoint paths for pairwise traffic, and supports low overhead and robust routing mechanisms. Additionally, in many online data-intensive services users tend to query data with more than one key, e.g., in Youtube video system users may want to find videos via both video ids and size ranges. Therefore, designing an indexing scheme for multi-dimensional query processing in HCN is useful and meaningful for real-world cloud applications, which has both theoretical and practical significance in this area. To search the data efficiently, the R-tree [23] based multi-dimensional index is used in our system. RT-HCN integrates HCN-based routing protocol and the R-Tree based indexing technology.

Similar to previous works, RT-HCN is a two-layer indexing scheme with a global index layer and a local index layer. Since datasets are distributed among different servers, we can use an R-Tree like indexing structure to index locally stored data for each server. Next, RT-HCN portionably distributes these local indices across servers as their global indices. To avoid single master server bottleneck, each server only maintains partial global index for its potential index range. Based on the characteristics of HCN, we design an index publishing rule to guarantee an "onto" mapping from global index to local stored data. We also propose the corresponding query processing algorithms to achieve query efficiency and load balancing for each node in the network. Finally, we prove theoretically that RT-HCN is both query-efficient and space-efficient, by which servers will not maintain redundant indices while a large number of users can concurrently process queries with low routing cost. We compare our design with

RT-CAN [7], a similar design for traditional P2P network. Experiments validate the efficiency of our proposed scheme and depict its potential implementation in data centers.

Our contribution of this paper is threefold: (1) to the best of our knowledge, we are the first to propose a distributed multi-dimensional indexing scheme for a specific DCN structure to improve query efficiency and system QoS; (2) noticing and taking advantage of the topology of HCN, we present a specialized mapping technique to improve global index allocation in the network, resulting query-efficiency and load-balancing for the cloud system; and (3) we theoretically prove the efficiency of RT-CAN, and compare our model numerically with RT-CAN [7], an indexing scheme for P2P network. Simulation results show that our scheme costs less index space for each node while provides faster query processing speed with higher bandwidth.

The rest of this paper is organized as follows. Section 2 summarizes the related work in this research area. Section 3 introduces the overview of our system, including coding of HCN and meta-servers. In Sec. 4 we illustrate the two-layer index construction with global index publishing rules. In Sec. 5 we depict the query processing algorithms with corresponding performance analysis theoretically. Section 6 discusses the maintenance and improvement of RT-HCN. Section 7 compares our design with the latest work RT-CAN in [7] and proves the efficiency of our construction. Finally, Section 8 provides a conclusion.

2 Related Works

Nowadays, there are lots distributed storage systems which assist to manage big data for cloud applications. Among them, we have excellent commercial implementations like key-value based system Amazon's Dynamo [2], Google's Google File System [1] (GFS), and BigTable [24], which aim at dealing with large scales of data. Meanwhile, some open source systems such as HDFS, HBase and HyperTable also provided a good platform for research use. Cassandra [3] is one non-rational database that combines features of BigTable and Dynamo. Some other systems such as Ceph [25], are designed to provide high performance in objects retrieval.

We want to build a second level overview index and our work follows the framework proposed in [8]. It offers an idea to build a two level index in cloud system for data retrieval on top of a physical layer. Moreover, an efficient and extensible framework for index in P2P based cloud system was put forward in [10]. However, more specialized topology has been designed to meet the requirement of today's cloud system and that is why we want to apply the two-level index design to specific data center network and discuss its improvement. As the topology of DCN is known, we can guarantee the processing time by calculating out the physical hops needed for a given query. While in P2P network only logical hops of the overlay network can be estimated.

Data center network (DCN) is the network infrastructure inside a data center, which connects a large number of servers via high-speed links and switches.

Compared to traditional cloud system which is usually based on P2P network, specially and carefully designed DCN topologies fulfill the requirements with low-cost, high scalability, low configuration overhead, robustness and energy-saving. DCN structures can be roughly divided into two categories, one is switch-centric such as VL2 [14] and Fat-Tree [13]. The other is server-centric like BCube [16], DCell [15], FiConn [17, 18], MDCube [19] and uFix [20]. They usually have more advantages than the former designs. HCN [22], the topology chosen in our system falls into the server centric topology. It is a well-designed network for data center and offers a high degree of regularity, scalability, and symmetry. Different from traditional P2P network, we have to be aware of the physical topology when we are discussing DCN and that is why we need to improve the mapping technique for distributing global index to fix a given network.

3 System Overview

In this section, we will introduce the topology of HCN and illustrate some basic features of it. Then, we explain some preliminary definitions for further illustration of index construction strategy.

3.1 Hierarchical Irregular Compound Network

HCN (Hierarchical irregular Compound Network) is a well-designed network for data center and offers a high degree of regularity, scalability, and symmetry. A level-h HCN with n servers in every single unit is denoted as HCN(n, h). HCN is a recursively defined structure. A high-level HCN(n, h) employs a low level HCN$(n, h-1)$ as a unit cluster and connects many such clusters by means of a complete graph. HCN$(n, 0)$ is the smallest module (basic construction unit) that consists of n dual-port servers and an n-port miniswitch. For each server, its first port is used to connect with the miniswitch while the second port is employed to interconnect with another server in different smallest modules for constituting larger networks. Figure 1 illustrates an example of HCN with $n = 4$ and $h = 2$, which consists of 64 servers. Each server is labeled according to a coding process, which will be introduced in Sec. 3.2.

It is easy to see that there is always multiple routes between any two servers in HCN and this is called multi-path routing which provides good features like high bandwidth, good balancing and error tolerance. This is the main aspect we want to concern during index construction and also becomes a main reason why special designed index scheme for specific network is well worth being discussed.

For clarity, we summarize the symbols with their meanings in Table 1. Some of them will be described in the following sections.

3.2 Meta-Server and Coding Strategy

Given an HCN$(4, h)$, there are 4^{h+1} servers in total and are coded by n ranging from 0 to $4^{h+1} - 1$. Thus we use S_n to denote the n^{th} server in the HCN.

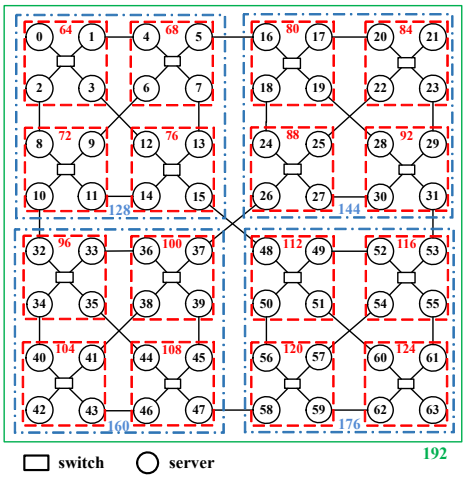


Fig. 1. HCN(4,2) with Coding for Meta-Server

Table 1. Symbol Description

Sym	Description	Sym	Description
n	Code for server and meta-server	h	The highest level of HCN
S_n	The server with code n	$\mathbf{B}$	Data boundary
M_n	The meta-server with code n	$\mathbf{B}_n$	Potential index range for M_n
$\mathbf{R}_n$	Representatives for M_n	R_n^i	The i^{th} representative for M_n
$\mathbf{N}_n$	S_n's node set for publishing.	N_n^i	The i^{th} publishing node for S_n

Since HCN itself is a recursively defined structure, there are also 4^{h-l} different HCN$(4,l)$'s $(0 \leq l \leq h)$ in the same HCN$(4,h)$. We consider each S_n and HCN$(4,l)$ $(0 \leq l \leq h)$ as a meta-server in our system.

Definition 1. *A Meta-server is a single server or an HCN$(4,l)$ $(0 \leq l \leq h)$ considered entirety.*

Meta-servers together constitute the overlay network and facilitate global queries, which is going to be explained in detail later. Meta-servers are also coded by n and denoted as M_n. The coding strategy of M_n is given by:

$$M_n = \begin{cases} S_n, & 0 \leq n < 4^{h+1} \\ \text{HCN}(4, q-1) \text{ consists of } S_r, S_{r+1}, \cdots, S_{r+4^q-1}, & n \geq 4^{h+1} \end{cases}, \quad (1)$$

where q and r from the above equation represent the quotient and the reminder when the given n is divided by 4^{h+1}. From the function, we know that M_n is equivalent to S_n when $n < 4^{h+1}$, and when $n \geq 4^{h+1}$ M_n represents a specific HCN$(4,l)$ $(0 \leq l \leq h)$. For example, Fig. 1 shows that in an HCN$(4,2)$, 64 meta-servers consist of only one server are coded with the number ranging from 0 to 63, 16 meta-servers formed by HCN$(4,0)$ are coded with 64, 68, $\cdots$, 124

(shown as red squares), 4 bigger meta-servers formed by HCN$(4, 1)$ are coded with number 128, 144, 160, 176 (shown as blue squares), while the biggest green square formed by the whole HCN$(4, 2)$ is coded with 192.

3.3 Representatives in Meta-Server

As is already mentioned, meta-servers form a higher-level overlay network and assist query processing in the network. However, as meta-servers are merely an abstract concept, we need to pick up several physical servers to be in charge of queries that are sent to corresponding meta-server from the overlay network.

Definition 2. *Representatives of a given meta-server are several carefully picked servers that physically deals with queries forwarded to that meta-server.*

We now provide our strategy for choosing representatives of a given meta-server M_n and there are two cases:

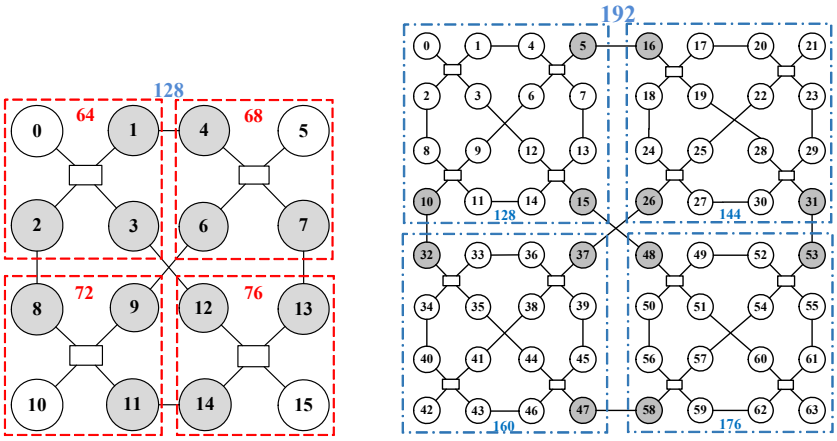
1. If $n < 4^{h+1}$, we chose S_n as the representative of M_n since $M_n = S_n$.
2. If $n \geq 4^{h+1}$, M_n, now, is actually an HCN$(4, l)$ $(0 \leq l \leq h)$, and we provide a special bit manipulation to find out its representatives. First we calculate the quaternary form of n, denoted as $q_0 q_1 \cdots q_m$. Here $m > h$ stands since $n \geq 4^{h+1}$. Secondly, we pick up the first $m - h$ bits as shown in Eqn. (2) and calculate the decimal number b. Then we replace each of the last b bits with q_*, where $q_* \neq q_{m-b}$ and will get several newly formed number. Finally we calculate the decimal form of the last $h + 1$ bits of the new numbers and servers coded with those numbers are the representatives for this M_n.

$$\overbrace{\underbrace{q_0 q_1 \cdots q_{m-h-1}}_{=b \text{ (decimal)}}}^{m-h \text{ bits}} \Big| \overbrace{q_{m-h} \cdots q_{m-b} \underbrace{q_{m-b+1} \cdots q_m}_{\text{replace part}}}^{h+1 \text{ bits}} \tag{2}$$

For example, the grey nodes in Fig. 2 (a), (b) illustrates the representatives for corresponding meta-server HCN$(4, l)$ when l equals to 0 and 1. It is obvious that these representatives actually takes the advantages of HCN topology and offer good connectivity which will facilitate query process.

We denote the representatives of M_n as set $\mathbf{R}_n$ and $\mathbf{R}_n$ has different number of entities in different situation. In case 1, $\mathbf{R}_n = \{R_n^1\}$ while in case 2, $\mathbf{R}_n = \{R_n^1, R_n^2, R_n^3\}$. Here R_n^i stands for the server which is the i^{th} representative of meta-server M_n, and R_n^i is called an l^{th}-level representative if and only if M_n is a meta-server formed by an HCN$(4, l)$ $(0 \leq l \leq h)$.

Now we give some explanation on choosing representatives. It is obvious to choose S_n as representatives for M_n when $n < 4^{h+1}$ because they are exactly the same. So we focus our discussion on case 2 here. According to the topology of HCN, it is not hard to find that all of the representatives we choose for M_n $(n \geq 4^{h+1})$ are servers that are more closely connected to other HCN$(4, l)$'s in the same level. Thus, our strategy cut the cost of queries forwarding and since there are three representatives for a single M_n, it also offers flexibility to choose the closet representative or the dullest one. This strategy also fits quite well with



(a). $l = 0$ (b). $l = 1$

Fig. 2. The Representatives for Meta-servers in HCN(4,2)

the multi-path routing of HCN. What's more, the following lemma and theorem show that our strategy also offers good scalability and balancing property.

Lemma 1. *Each meta-server M_n ($n \geq 4^{h+1}$) has exactly three representatives.*

Proof: There are four different bits in quaternary number and our strategy replace the bits with a same bit different with q_{m-b} in Eqn. (2), so the result is three. The biggest $M_{(h+1)\cdot 4^{h+1}}$ is the only special one that has four representatives since the bit q_{m-b} does not exist in this case. □

Theorem 1. *Each server S_n will be the representative for exactly two different meta-servers.*

Proof: Firstly, it is obvious that each server should be chosen as a representative for M_n where $n < 4^{h+1}$. Secondly, according to Lemma 1, the representative-time in the whole system is $2 \cdot 4^{h+1}$ in total. Thirdly, no server can be selected for three times. Since the quaternary form of the coding for a given server is fixed, there exist only one way for it to be split according to the form given by Eqn. (2). Combining the above discussion, we can draw a conclusion to our proof according to the Pigeonhole Principle. □

4 Indexing Construction

In this section, we first introduce the vital component of the higher level overlay network, which is essential for constructing the global index. Then we introduce the construction our two-layer index in detail with index publishing rules.

4.1 Potential Indexing Range

Before we begin our discussion of index construction, we illustrate another essential concept about our meta-servers. In order to construct the global index for multi-dimensional data in our overlay network, we have assigned each meta-server a potential indexing range. Thus, for any given queries, we can figure out which meta-server is responsible for it and then the query is processed by the representatives of that meta-server.

The multi-dimensional data forms a data boundary denoted as $\mathbf{B}$, which is a k-dimensional rectangle as the bounding box of the spatial data objects:

$$\mathbf{B} = (B_0, B_1, B_2, \cdots, B_k) \ . \tag{3}$$

Here k is the number of dimensions and each B_i is a closed bounded interval $[l_i, u_i]$ describing the extent of the data along dimension i. To better illustrate our idea, the following discussion focuses on an example situation when $k = 2$.

Definition 3. *Potential indexing range is an abstract attribute assigned to meta-server and indicates which meta-server (indeed the subordinate representatives) is (are) responsible for processing a coming query.*

Suppose the data is bounded by $\mathbf{B} = (B_0, B_1)$, where B_0 is $[l_0, u_0]$ and B_1 is $[l_1, u_1]$. We calculate a quaternary number Q_n for each meta-server M_n and use it to help figure out the potential index range $\mathbf{B}_n$ for M_n. Suppose $q_0 q_1 \cdots q_m$ is the corresponding quaternary form for n, then Q_n is calculated according to the following two cases:

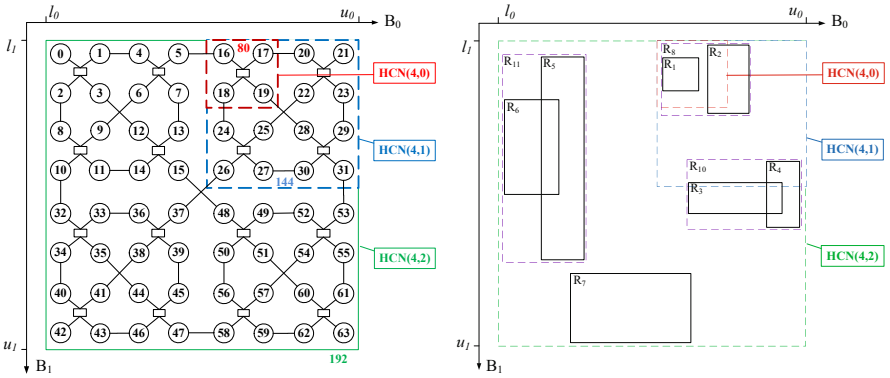
1. If $m \leq h$, we add $m - h$ consecutive 0's to the front of $q_0 q_1 \cdots q_m$ and construct an $h + 1$ bit quaternary number $Q_n = 00 \cdots 0 q_0 q_1 \cdots q_m$.
2. If $m > h$, $q_0 q_1 \cdots q_m$ can be split as the form explained in Eqn. (2). In this situation, we pick out the last $h + 1$ bits and delete the replace part to get $Q_n = q_{m-h} q_{m-h+1} \cdots q_{m-b}$.

Now, we use the following iteratively defined function to calculate $\mathbf{B}_n$.

$$\mathbf{B}_n = pir(\mathbf{B}, Q_n) = \begin{cases} pir(([l_0, \frac{l_0+u_0}{2}], [l_1, \frac{l_1+u_1}{2}]), Q'_n), & q_0 = 0 \\ pir(([\frac{l_0+u_0}{2}, u_0], [l_1, \frac{l_1+u_1}{2}]), Q'_n), & q_0 = 1 \\ pir(([l_0, \frac{l_0+u_0}{2}], [\frac{l_1+u_1}{2}, u_1]), Q'_n), & q_0 = 2 \\ pir(([\frac{l_0+u_0}{2}, u_0], [\frac{l_1+u_1}{2}, u_1]), Q'_n), & q_0 = 3 \\ ([l_0, u_0], [l_1, u_1]) & Q_n = \varnothing \end{cases} , \tag{4}$$

where Q_n is a quaternary number denoted as $q_0 q_1 \cdots q_i$ and Q'_n is a newly constructed quaternary number $q_1 q_2 \cdots q_i$. For higher dimensional data, the data range can be seen as a hyper-rectangle and the potential index ranges for different meta-servers are sequenced according to their coding in a similar way.

For example in Fig. 3 (a), two axes B_0 and B_1 denote the range of data in two dimensional space (w.l.o.g., we assume it generates a square area, rather than a rectangle). Then the potential indexing range for M_{80} (denoted as red square) should be ($[\frac{l_0+u_0}{2}, \frac{l_0+3u_0}{4}]$,$[l_1, \frac{3l_1+u_1}{4}]$); the *pir* for M_{144} (denoted as blue square) is($[\frac{l_0+u_0}{2}, u_0]$, $[l_1, \frac{l_1+u_1}{2}]$), while the *pir* for M_{192} is ($[l_0, u_0]$, $[l_1, u_1]$).



(a). Potential Index Range (b). Local R-tree Index Nodes

Fig. 3. Illustration of Range Mapping and Index Distribution

4.2 Global Index in Overlay Network

As is discussed above, each data server S_n has built an R-tree index for its local data to facilitate multi-dimensional search. Then, S_n adaptively selects a set of index nodes $\mathbf{N}_n = \{N_n^1, N_n^2, \cdots, N_n^{d_n}\}$ from its local R-tree and publishes each N_n^i to the representatives of a specific meta-server whose potential indexing range just covers the minimum bounding range of N_n^i. We initially choose the last but one level index nodes from a given R-tree to be published since these nodes are usually not frequently updated and do not introduce too many false positives either. The format of the published R-tree nodes is (n, mbr), where n indicates the origin server for storing the data and mbr is the minimal bounding range of the published R-tree node. After receiving the published nodes, representatives buffers the index in memory. In this way, the global index composes of several R-tree nodes from the local indexes and is distributed over the data center.

Algorithm 1. Index Publishing (For S_n)

```
1 Nn =getSelectedRTreeNode(Sn)
2 for each N_n^i ∈ Nn do
3 |   Find the least n' s.t. B_n' fully covers N_n^i.mbr
4 |   Get the representatives R_n' for M_n'
5 |   for each S_k ∈ R_n' do
6 |   |   S_k inserts (n, N_n^i.mbr) into its global index set
7 |   end
8 end
```

Fig. 3 (b) provides a simple example of a local R-tree. If the node R_1 is selected to be published, it should be published to server S_{17}, S_{18}, S_{19}, which are representatives of the HCN$(4, 0)$ shown in red. Similarly, if the node R_3 is

selected to be published, it should be published to server S_{16}, S_{26}, S_{31}, which are representatives of the HCN$(4, 1)$ shown in blue.

The average routing cost for one-to-one traffic in HCN is $O(2^{h+1})$ [22], and the cost for information transmission between representatives of the same meta-server is $o(2^{h+1})$, thus the cost to publish an index node should be $O(2^{h+1})$ on average, and it equals to $O(\sqrt{N})$ when $n = 4$, where N is the total number of servers in the given HCN. For more general situation, the cost is given by $N^{-\log_2 n}$ and can be reduced as n get larger. We also want to mention here that the cost shown above is exactly physical hops between servers while previous works claims that it takes only $O(\log N)$ to publish index in P2P network but they are only discussing hops in the overlay network. Since the physical connection of P2P network is unclear, the physical hops can be hard to exam and constrain. Another improved feature for index publishing in our system is that under this design we can make sure that each index node is published to exactly only three servers in the system. However, the strategy used in [7] publish the nodes to other servers as long as the range of the index node overlaps with the potential index range of the given servers. Then, the amount of index nodes published in the system is hard to control.

5 Query Processing

In this section we show how our global index can be applied to process queries and reduce the cost of physical hop counts in an HCN.

5.1 Point Query

A query is denoted as $Q(value)$ where $value = (v_0, v_1)$, indicating a multi-dimensional data point. Given an indexed R-tree node N_i^n, N_i^n should be searched to retrieve possible results if and only if $N_i^n.mbr$ covers the query point (v_0, v_1). Such node N_i^n, however, can be published to several different representatives in our system. The following theorem shows that we have to search a set of servers' global index to get the full results. Our design splits the processing of a query into two phases. In the first phase, the query is done by the response server S_{n_1} of the meta-server that is responsible for the query and the representatives look up the global index, search the buffered R-tree nodes and return the entries that satisfy the query. In the second phase, based on the received index entries, the query is forwarded to the corresponding physical server S_{n_2} and retrieve the results via the local R-tree.

Theorem 2. *The published index node that may contain results for a given value (v_0, v_1) are in representatives of $h + 2$ meta-servers. Moreover, exactly $h + 1$ servers need to be searched in total to get the full results.*

Proof: Suppose an indexed R-tree node N_i^n contains the search value (v_0, v_1), it is certain that the destination for publishing this index node must be the representative of some meta-servers whose potential index range covers point (v_0, v_1),

other wise it will contradict with our publishing rule. Therefore, we can easily figure out there are $h+2$ different meta-servers that meet the above condition. They respectively consist of a single server, an HCN$(4,0)$, an HCN$(4,1)$, $\cdots$, and the biggest HCN$(4,h)$. Since the meta-server formed by a single server takes itself as a representative and is also one of the representatives for the meta-server formed by HCN$(4,l)$ that meet the above condition for some l, the total number of servers to search is then $h+1$. It is also good to notice that the $h+1$ servers are actually representatives of different levels that is from 0^{th} level to h^{th} level, this will facilitate our explanation for query forwarding. □

Although it seems that $h+1$ is a quite big number while the total number of servers are 4^{h+1}, we provide our strategy for forwarding the query based on a greedy algorithm and show that the cost of forwarding is really small even when there are $h+1$ servers to be searched. As we have discussed above, there are $h+1$ representatives to search and they range from level 0 to level h. To process a query $Q(v_0, v_1)$, we first forward the query to the nearest h^{th} level representative S_{n_h} which is responsible for the given query. S_{n_h} searches its buffered global index and returns matched results to the user. Then, it forwards the query to the nearest $h-1^{th}$ level representative $S_{n_{h-1}}$, respectively. Until a 0^{th} level representative has been searched, we can finally get the full results. The head of the routing message follows the format of $\{l, value\}$, where l is the level for representative and $value$ is the querying point. Alg. 2 describes such process in detail.

Algorithm 2. Point Query Processing

Input : $\{l, value\}$
Output: **N** (the result set of indexed R-tree nodes)

```
1 N = ∅
2 for ∀N ∈ S.globalIndex do
3 |   if N.mbr covers value then
4 |   |   Add N into N
5 |   end
6 end
7 S_next = the nearest l − 1^th level representative for value
8 Forward query message {l − 1, value} to S_next
9 return N
```

We can see from the above algorithm that the shortest query path for a given query is $O(\log N)$ and if we want to maintain the multi-path routing of HCN, our index publishing scheme does bring out some extra cost for specific queries. But since the cost of forwarding queries from any $l+1^{th}$ level representative to the nearest l^{th} level representative will never exceed the maximum cost of point-to-point routing in an HCN$(4,l)$ which is $O(2^{l+1})$, and noticing that under this design only with a probability of $\frac{1}{4}$ will a query pass result in redundant routing, thus, the average cost of query in our system will only be $\frac{5}{4}$ times of the cost

of point-to-point routing in an HCN$(4, h)$ and then is still constrained by a logarithmic scale. What's more, with a similar analysis, it is easy to figure out that the parameter $\frac{5}{4}$ will decrease to $\frac{n+1}{n}$ when considering an HCN(n, h). We mention again that the cost we were discussing above are all physical connections rather than hops in the overlay network.

5.2 Range Query

A range query is denoted as $Q(range)$, where $range = ([l_0, u_0], [l_1, u_1])$ is a multi-dimensional hypercube. If an R-tree node's boundary $N_i^n.mbr$ overlaps with *range*, it should be searched to retrieve the possible query results. Let us think about an easy case first, if the *range* is small enough and can be fully covered by some $\mathbf{B}_n$, where n is smaller than 4^{h+1}, then $Q(range)$ actually goes in the same way we do for point queries in the last subsection. Inspired by this special case, we can figure out that range query is an variation of point query, with only a small modification.

We can first find out the smallest HCN$(4, l)$ that fully covers the queried range, then just as what we did in point query, we first forward the query to the nearest h^{th} level representative S_{n_h}, which is responsible for the given range. Then, S_{n_h} forwards the query to the nearest $h - 1^{th}$ level representative $S_{n_{h-1}}$, respectively, until an l^{th} level representative has been searched. Then, different from what we did in point query, from the l^{th} level on, the representatives will forward the query to all the lower level representatives, whose potential index range overlaps with *range*, to guarantee the correctness and completeness of results. Since routing cost in HCN$(4, l)$ where $l < h$ is $o(\log N)$, with similar analysis for point query, it is easy to figure out that the cost for a range query is still $O(\log N)$, and the detail of the range query algorithm is omitted here as it is similar to Alg. 2.

6 Further Discussion and Optimization

We only used HCN$(4, 2)$ and 2-dimensional data for discussion in the above sections, this does not mean these two parameters should be fixed. When considering the implementation of an HCN(n, h), the function to map each meta-server a different potential index range should be changed since n is different. Noticing that h is a parameter that indicates the scalability of the whole system, it does not affect much the design of the mapping function because the system is a recursively designed architecture. In a situation where $n \neq 4$, we can also build a function similar to Equation (4), the only difference is that we are now dealing with an n-dimensional number rather than a quaternary number. For higher dimensional data, we can take advantage of the space-filling curve [27] to reduce dimension. Taking advantage of these space filling curves, it is possible to generate one-to-one mappings between spaces of different dimensions, that is, we can apply our strategy similarly to higher-dimension data.

We also offer an optimization for data distribution. It is sure that the system will be more balanced if the data with close features are allocated closer. It is

also good to keep the locality coherence of data for retrieval, especially for multi-dimensional data. Locality sensitive functions offered by [26] can facilitate data distribution and offer good performance of query processing. Another significant improvement can be done is about queries with specific frequency. It is a common scene in real world that some data are more hot and queried more frequently than others. In this case, it is sure that some servers which are responsible for the hot data are more busy and are in the risk of being bottleneck for the system. A heuristic strategy for this problem is that we may want to assign a dynamic potential index range for each server and discuss the cost of maintenance and the improvement of performance.

7 Performance Evaluation and Numerical Experiments

In this section, we evaluate the performance of RT-HCN. We simulate an HCN$(4, 2)$, generate the multi-dimensional data and randomly allocate them in different servers. We consider randomly generated data together with data following Zipf distribution to prove the scalability of RT-HCN. The queries including both point queries and range queries are randomly generated from any server. Since the cost of query processing is proved to takes only $N^{-\log_2 n}$ physical hops for any given query, performance of our system discussed in this section focus mainly on two metrics: Heat of each server and Index nodes in each server, both of which reflects the high scalability and balance of the system. Table 2 is the parameters used in our experiments.

Table 2. Parameters Used in Our Experiments

Parameter	Default	Range	Meaning
D	320000	[320000, 1280000]	Number of data items in total
h	2		Highest level of HCN
N	64		Total number of servers

First, we estimate the number of published index nodes in different servers in the network. We compare the results of our publishing strategy with the one used in RT-CAN to show that we provide a more scalable and balanced publishing rule. Our evaluation is first carried out with randomly generated data and then with data that obey centralized distribution. Fig. 4 and Fig. 5 show the data distribution in the experiments.

We then vary the number of data items distributed to each server and investigate the changes in number of published index nodes. Fig. 6-8 show the number of global indices on each server according to RT-HCN and RT-CAN publishing rules, where number of data item varies from $320,000$ to $1,280,000$. From the results we can see that our system maintains less index entries at each server compared to RT-CAN under uniform data distribution. Similarly, results in Fig. 9-11 show that when data are not randomly distributed, our design still provides a better result with tolerate number of indices.

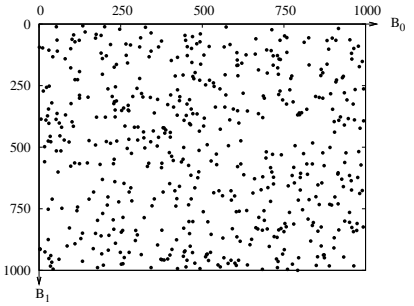


Fig. 4. Uniform Data Distribution

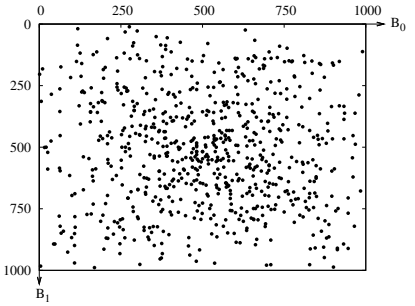


Fig. 5. Zipf Data Distribution

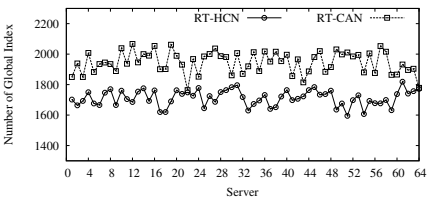


Fig. 6. D=320,000

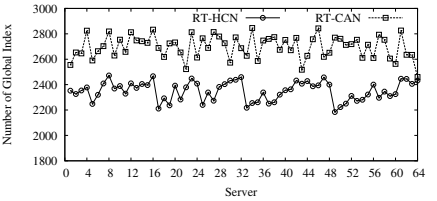


Fig. 7. D=640,000

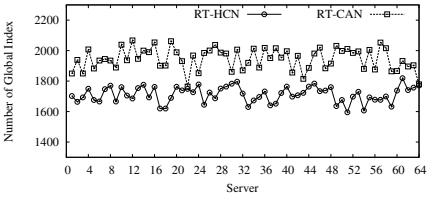


Fig. 8. D=320,000

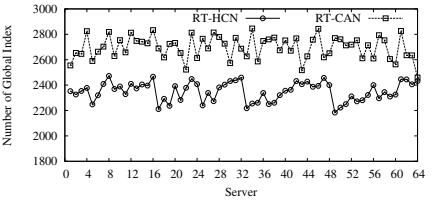


Fig. 9. D=640,000

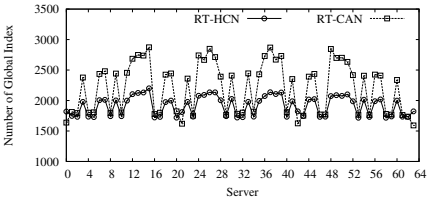


Fig. 10. D=640,000

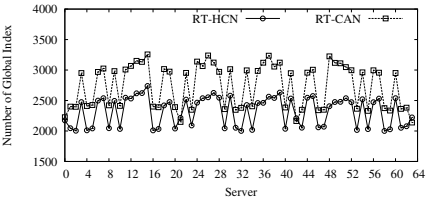


Fig. 11. D=1,280,000

Finally, we estimate the visiting frequency of different servers in the network (denoted as a heatmap) when a given number of randomly generated queries are processed. The heatmap is drawn according to the access frequency of each server and directly reflects the processing condition of servers in the system. When a multiple-paths routing is applied, the variance of access frequency of the whole system is reduced, indicating a more balancing performance. Moreover, when a larger HCN is built, good balancing performance of smaller HCNs reduces the risk of any server being the bottleneck. Thus, in other words, good results from heatmap also reflect good scalability of our system design. The results are shown in Fig. 12 and Fig. 13. We can see that shortest path routing offers a not bad performance since most servers are almost as busy as each other and when we apply multiple paths for one-to-one traffic we get a even more balanced performance since our publishing strategy is not locality based.

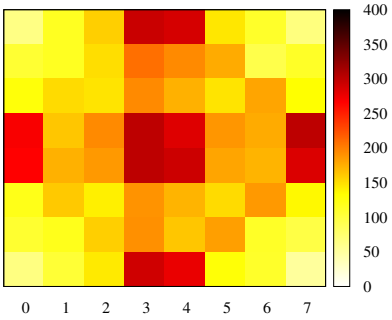


Fig. 12. Heatmap Shortest-Paths

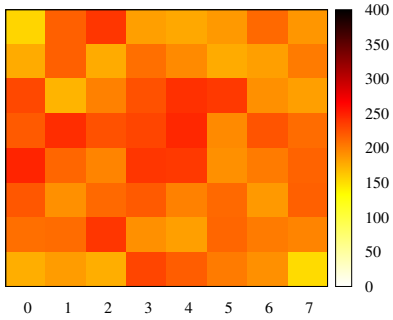


Fig. 13. Heatmap Multiple-Paths

8 Conclusion

In this paper, we proposed an indexing scheme named RT-HCN for multi-dimensional query processing in data centers, which are the infrastructures for building cloud storage systems and are interconnected using a specific data center network (DCN). RT-HCN is a two-layer indexing scheme, which integrates HCN-based routing protocol [22] and the R-Tree based indexing technology, and is portionably distributed on every server. Based on the characteristics of HCN, we design a special index publishing rule and query processing algorithms to guarantee efficient data management for the whole network. We prove theoretically that RT-HCN is both query-efficient and space-efficient, by which each server will only maintain a constrained number of indices while a large number of users can concurrently process queries with low routing cost. We compare our design with RT-CAN [7], a similar design for traditional P2P network. Experiments validate the efficiency of our proposed scheme and depict its potential implementation in data centers.

References

1. Ghemawat, S., Gobioff, H., Leung, S.-T.: The Google file system. ACM SIGOPS 37(5), 29–43 (2003)
2. DeCandia, G., Hastorun, D., Jampani, M., et al.: Dynamo: amazon's highly available key-value store. ACM SIGOPS 41(6), 205–220 (2007)
3. Lakshman, A., Malik, P.: Cassandra: a decentralized structured storage system. ACM SIGOPS 44(2), 35–40 (2010)
4. Beaver, D., Kumar, S., Li, H.C., et al.: Finding a Needle in Haystack: Facebook's Photo Storage. In: USENIX OSDI, pp. 47–60 (2010)
5. Baker, J., Bond, C., Corbett, J.C., et al.: Megastore: Providing Scalable, Highly Available Storage for Interactive Services. ACM CIDR 11, 223–234 (2011)
6. Corbett, J.C., Dean, J., Epstein, M., et al.: Spanner: Google's globally-distributed database. ACM TOCS 31(3), 8 (2013)
7. Wang, J., Wu, S., Gao, H., et al.: Indexing multi-dimensional data in a cloud system. In: ACM SIGMOD, pp. 591–602 (2010)
8. Wu, S., Wu, K.-L.: An Indexing Framework for Efficient Retrieval on the Cloud. Bulletin of TCDE of the IEEE Computer Society 32(1), 75–82 (2009)
9. Wu, S., Jiang, D., Ooi, B.C., Wu, K.-L.: Efficient b-tree based indexing for cloud data processing. ACM VLDB 3(1-2), 1207–1218 (2010)
10. Chen, G., Vo, H.T., Wu, S., et al.: A Framework for Supporting DBMS-like Indexes in the Cloud. ACM VLDB 4(11), 702–713 (2011)
11. Jagadish, H.V., Ooi, B.C., Vu, Q.H.: Baton: A balanced tree structure for peer-to-peer networks. In: ACM VLDB, pp. 661–672 (2005)
12. Ratnasamy, S., Francis, P., Handley, M., et al.: A scalable content-addressable network. ACM SIGCOMM 31(4), 161–172 (2001)
13. Al-Fares, M., Loukissas, A., Vahdat, A.: A scalable, commodity data center network architecture. ACM SIGCOMM 38(4), 63–74 (2008)
14. Greenberg, A., Hamilton, J.R., Jain, N., et al.: VL2: a scalable and flexible data center network. ACM SIGCOMM 39(4), 51–62 (2009)
15. Guo, C., Wu, H., Tan, K., et al.: Dcell: a scalable and fault-tolerant network structure for data centers. ACM SIGCOMM 38(4), 75–86 (2008)
16. Guo, C., Lu, G., Li, D., et al.: BCube: a high performance, server-centric network architecture for modular data centers. ACM SIGCOMM 39(4), 63–74 (2009)
17. Li, D., Guo, C., Wu, H., et al.: FiConn: Using backup port for server interconnection in data centers. In: IEEE INFOCOM, pp. 2276–2285 (2009)
18. Li, D., Guo, C., Wu, H., et al.: Scalable and cost-effective interconnection of data-center servers using dual server ports. IEEE/ACM TON 19(1), 102–114 (2011)
19. Wu, H., Lu, G., Li, D., et al.: MDCube: a high performance network structure for modular data center interconnection. In: ACM CoNEXT, pp. 25–36 (2009)
20. Li, D., Xu, M., Zhao, H., Fu, X.: Building mega data center from heterogeneous containers. In: IEEE ICNP, pp. 256–265 (2011)
21. Guo, D., Chen, T., Li, D., et al.: BCN: expansible network structures for data centers using hierarchical compound graphs. In: IEEE INFOCOM, pp. 61–65 (2011)
22. Guo, D., Chen, T., Li, D., et al.: Expandable and cost-effective network structures for data centers using dual-port servers. IEEE Trans. Comp. 62(7), 1303–1317 (2013)
23. Antonin, G.: R-trees: A dynamic index structure for spatial searching. ACM SIGMOD 14(2), 47–57 (1984)

24. Chang, F., Dean, J., Ghemawat, S., et al.: Bigtable: A distributed storage system for structured data. ACM TOCS 26(2), 4:1–4:26 (2008)
25. Weil, S.A., Brandt, S.A., Miller, E.L., et al.: Ceph: A scalable, high-performance distributed file system. In: USENIX OSDI, pp. 307–320 (2006)
26. Charikar, M.S.: Similarity estimation techniques from rounding algorithms. In: ACM STOC, pp. 380–388 (2002)
27. Sagan, H.: Space-filling curves. Springer, New York (1994)