# A Study on Multi-Attribute Database Indexing on Cloud System[1]

Yu-Lung Lo and Choon-Yong Tan[2]

***Abstract* — Recent years, the Cloud computing technologies have become more and more important for many existing database applications. The Cloud platform provides an ease to use interface between providers and users, allow providers to develop and provide software and databases for users over locations. Currently, there are many Cloud platform providers support large-scale database services. However, most of these Cloud platform architectures only support simple keyword-based queries and can't response complex query efficiently due to lack of efficient in multi-attribute index techniques. Existing Cloud platform providers seek to improve performance of indexing techniques for complex queries. In this paper, we evaluate the existing multi-attribute indexing structures for Cloud platform. We conclude our experimental results to suggest the more efficient and scalable multi-attribute index structure for Cloud platform.**

***Index Terms* — Cloud computing, multi-attribute index, R-tree, *k*-d tree**

## I. INTRODUCTION

The Cloud computing is an emerging business solution. It can address the requirements of each software service to distribute the storage space and all kinds of the service on the resource pool. The user does not need to purchase any hardware or software and flexible to upgrade amount of resource according their own actual demand from provider. The Cloud system generated business opportunity and future trend in the software industry. According to estimation from Merrill Lynch [14], by 2011, the profit of Cloud computing market should reach $160 billion, including $95 billion in business and $65 billion in online advertising. Due to the commercial potential of the Cloud platform, more IT companies are increasing their investments in Cloud develop. Existing Cloud infrastructures include Amazon's Elastic Computing Cloud (EC2) [15], IBM's Blue Cloud [12] and Google's Map Reduce [6]. Luis M. Vaquero etc. [18] listed up to 22 definitions and analyses about Cloud computing. There are ten characteristics of Cloud computing in their summaries: user friendliness, virtualization, Internet centric, variety of resources, automatic adaptation, scalability, resource optimization, pay-per-used, service SLAs (Service-Level Agreements) and infrastructure SLAs [9].

The Cloud computing is a virtual computation resource which may maintain and manage by itself, normally for a lot of large-scale server cluster structures including computation servers, storage servers, the bandwidth resources and so on [21]. Cloud platform compose by a number of computer resources and store a large number of data, and provide services to millions of global user. Resource allocation usually computes in Cloud platform and make user feel that owns personal infinite resources. Providing scalable database services is one of most important issue for extending many applications of the Cloud platform. The Cloud platform simplifies to provide a large-scale distributed database system however performing indexing and searching in such a database on the Cloud platform has become new challenges to realize.

The traditional distributed system structure lacks of scalability and reliability therefore it cannot be directly applied to the new platform. Due to the diversity of applications, database services on the Cloud must support large-scale data analytical tasks and high concurrent On-Line Transaction Processing (OLTP) queries. When unexpected large searching enquiries occur, it may happen that users meet the situation of out of supported by system resource and disable of quality of service [23]. However, currently the Cloud platform only supports simple keyword-based queries. It can't answer complex queries efficiently due to lack of efficient index techniques. There were few research reports proposed indexing schemes for Cloud platform to manage the huge and variety data. These schemes create global index for master nodes and local index for each slave (or storage) node as shown in Figure 1. To prevent the bottleneck, the global index is distributed and maintained in several master nodes. The local index manages the local data in a slave node for local data search and the global index manages the tree node in local index for searching entries of slave nodes. All these index structures are based on existing index structures, such as R-tree[10] and *k*-d tree[7], which can support multi-attribute/multi-dimensional indexing or spatial indexing. In this paper, we would like to survey and evaluate the existing multi-attribute index schemes then provide a suggestion for applying multi-attribute indexing in Cloud platform.

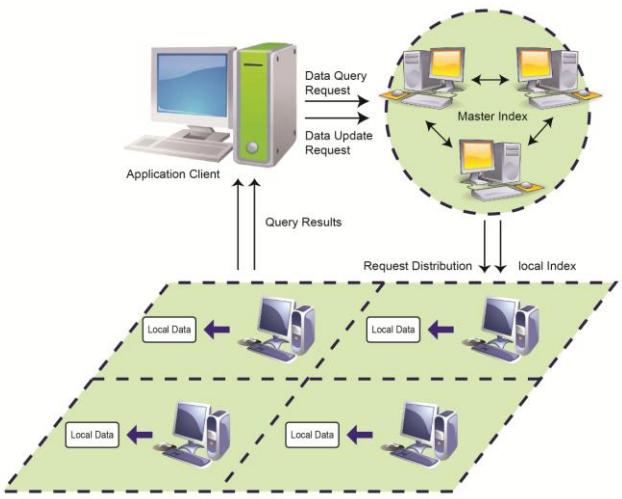


Figure 1. Index structure for Cloud platform

[1] This work was supported by National Science Council of ROC Grant NSC 100-2221-E-324-027.

[2] Yu-Lung Lo and Choon-Yong Tan are with the Department of Information management, Chaoyang University of Technology, Wufong District, Taichung, 41349 Taiwan. Phone: +886-04-2332-3000 ext. 7121; fax: +886-4-2374-2337; e-mail: yllo@cyut.edu.tw; s9914644@cyut.edu.tw.

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

The remaining of this paper is organized as follows: in section 2, we review the existing multi-attribute indexing schemes for Cloud platform. After that, we design experiments to evaluate the existing multi-attribute indexing structures in section 3. Finally, a conclusion and our suggestion are given in the last section.

## II. RELATED WORK

There are several distributed storage systems to manage large amounts of data such as Google File System (GFS) serves Google's applications with large data volume [8], BigTable is a distributed storage system for managing structured data of very large scales [4], and PNUTS proposed by Yahoo is a hosted and centrally controlled parallel and distributed database system for Yahoo's applications [5]. These systems organize data into chunks then disseminate chunks into numbers of clusters to improve data access parallelism.

The concept of Cloud computing evolves from internet search engines' infrastructure. The differences between Cloud computing and DBMS are that the Cloud computing does not adopt order-preserving tree indexes, such as B-tree or hash table [9]. Aguilera et al. [1] proposed a fault-tolerant and scalable distributed B-tree for their Cloud systems. Although B-tree has been widely used as single attribute index in database systems, it is inefficient in dealing with indices composed of multi-attributes [23]. To improve the weakness of Cloud computing, to build a multi-attribute index may support more types of queries on Cloud computing platforms. Therefore, Xiangyu Zhang et al. proposed an Efficient Multi-dimensional Index with Node Cube for Cloud computing system [23] and Jinbao Wang et al. built the RT-CAN index in their Cloud database management system in 2010 [19]. Both these two schemes are based on *k*-d tree and R-tree. The brief introductions for these two schemes as well as *k*-d tree and R-tree are as follows.

### A. Effecent Multi-dimensional Index with Node Cube

In 2009, Xiangyu Zhang et al. [23] proposed an efficient approach to build multi-dimensional index for Cloud computing system. In this approach, they build local *k*-d tree index for each slave nodes due to *k*-d tree can efficiently support point query, partial match query and range query. To prune irrelevant nodes on query processing, they construct a node cube for each slave node. A node cube indicates the range of value on each indexed attribute in this node. After they build a cube for each slave node, they maintain the cubes on master nodes with an R-tree. The reason of choosing R-tree for cube information is that the R-tree was designed for managing data regions and in their scenario the cubes are multi-dimensional data regions. They call this index approach EMINC: Efficient Multi- dimensional Index with Node Cube as shown in Figure 2.

With the node cube information in EMINC, query processing can be improved by pruning irrelative nodes in the nodes locating phase. And in order to keep cube information available and useful, insertion and deletion on slave nodes that may change their cubes should inform master nodes for update of cube.

However, EMINC has some limitations and under some occasions, the performance could still be poor. The authors extend EMINC to use multiple node cubes to represent a slave node in which data records on one slave node will be represented by multiple node cubes. The shape and amount of node cubes is dependent on the method used for cutting the original single node cube.

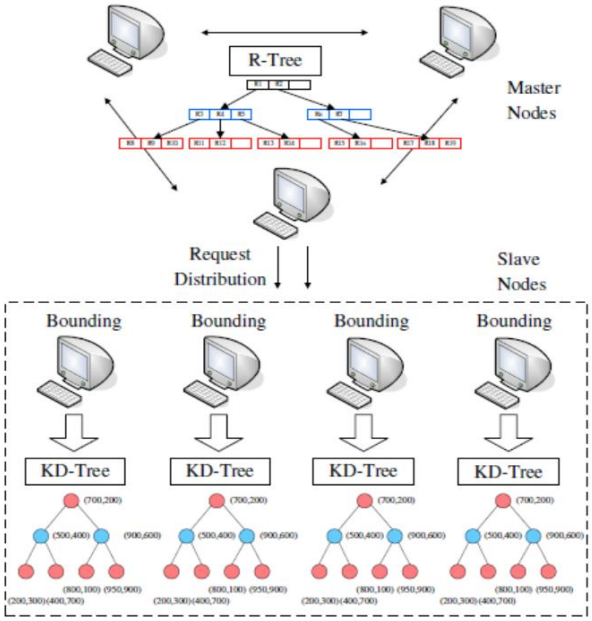


Figure 2. Framework of EMINC [23]

### B. RT-CAN Index

The RT-CAN is a multi-dimensional indexing scheme proposed by Jinbao Wang et al. in 2010 [19]. RT-CAN integrates CAN-based routing protocol [17] and the R-tree based indexing scheme to support efficient multi-dimensional query processing in a Cloud system.

CAN (Content Addressable Network) [17] is a scalable, self-organized structured peer-to-peer overlay network. The RT-CAN index is built on a shared-nothing cluster, where application data are partitioned and distributed over different servers. In this approach, the global index composes of some R-tree nodes from the local indexes and is distributed over the cluster. The global index can be considered as a secondary index on top of the local R-trees. This design splits the processing of a query into two phases. In the first phase, the processor looks up the global index by mapping the query to some CAN nodes. These CAN nodes search their 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 storage nodes, which retrieve the results via the local R-tree. The index structure and data service of RT-CAN is shown in Figure 3.

### C. k-d Tree

The *k*-d tree (short for *k*-dimensional tree) was proposed by Jon Louis Bentley in 1975 [3]. The *k*-d tree is a space-partitioning data structure for organizing points in a k-dimensional space. For example, the definition of a 2-d tree is a binary tree satisfying the following two conditions: (with root a level 0)

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

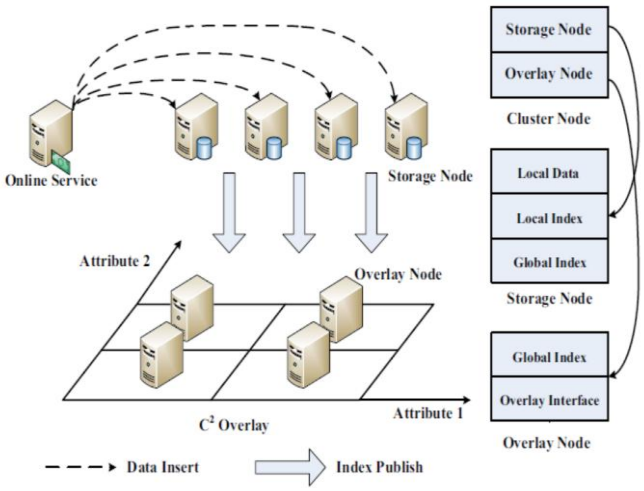


Figure3. Data Service of RT-CAN Index [19]

1. For node *N* with *level*(*N*) is even, then every node *M* under *N.llink* has the property that *M.xval* < *N.xval*, and every node *P* under *N.rlink* has the property that *P.xval* ≧ *N.xval*.
2. For node *N* with *level*(*N*) is odd, then every node *M* under *N.llink* has the property that *M.yval* < *N.yval*, and every node *P* under *N.rlink* has the property that *P.yval* ≧ *N.yval*.

Where *xval* and *yval* denote the coordinates of *x* and *y*, respectively; and *llink* and *rlink* are the pointers to the left child node and right child node, respectively. For instance, a two-dimensional space consists of some data points as shown in Figure 4. Such that we can create a 2-d tree for these data points as shown in Figure 5, and the space is partitioned into Figure 6.

The three and more dimensional *k*-d trees can be derived in the similar way. Furthermore, adaptive *k*-d tree [11] was proposed by Andreas Henrich et al. in 1989, when the *k*-d trees are established, according to the data feature, a specific data plane of division is selected to make each division into two equal value subspaces, but this optimization is only useful for static data. For dynamically updated data, the tree structure needs to be completely reorganized.

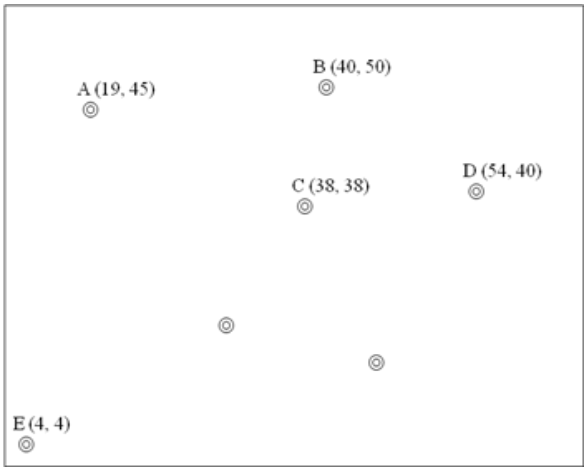


Figure 4. 2-d space with data points

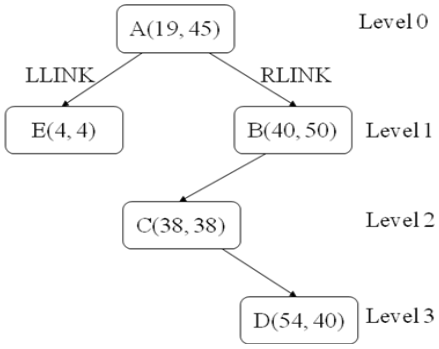


Figure 5. an example of 2-d tree

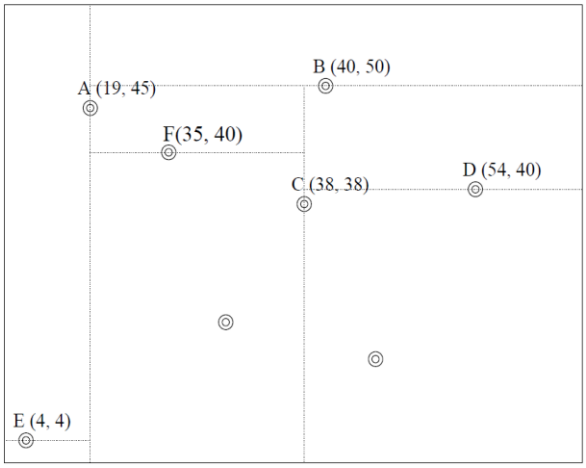


Figure 6. space partitioned by 2-d tree

### *D. R-tree*

The R-tree was proposed by Antonin Guttman in 1984 [10]. R-tree is a tree data structure used for spatial access methods. It groups nearby objects and represents them with their minimum bounding *d*-dimensional rectangle in the next higher level of the tree. Each node of the R-tree corresponds to the minimum bounding *d*-dimensional rectangle that bounds its children. Since all objects lie within this bounding rectangle, a query that does not intersect the bounding rectangle can also not intersect any of the contained objects. In another words, R-tree uses the bounding boxes to decide whether or not to search inside a sub-tree. At the leaf level, each rectangle describes a single object; at higher levels the aggregation of an increasing number of objects. R-tree is a balanced search tree which organizes the data in pages and is designed for storage on disk.

In an R-tree for two-dimensional space, it has an associated order *k* and each non-leaf node contains a set of at most *k* rectangles and at least $\lceil k/2 \rceil$ rectangles. For example, there are three rectangles regions containing nine objects as shown in Figure 7. An R-tree for this 2-d space can be created as in Figure 8.

Furthermore, a 3D R-tree, proposed in [22], considers time as an extra dimension and represents 2D rectangles with time intervals as three-dimensional boxes. This tree can be the original R-tree.

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

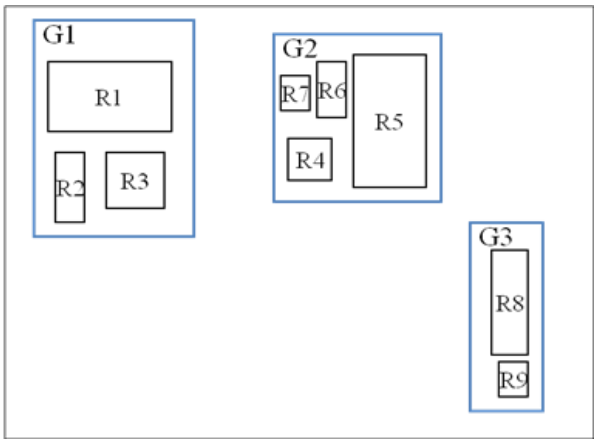


Figure 7. rectangles for 2-d space

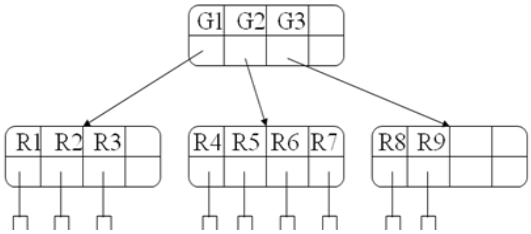


Figure 8. R-tree for 2-d space

## III. EXPERIMENTAL

Since the most existing multi-attribute index structures for Cloud platform are based on *k*-d tree and R-tree, and there is without study for evaluating both schemes yet, we would like to examine these two index structures in this section. In our experiment, we prepared infrastructure including machines which are connected together to simulate Cloud computing platforms. Each machine had a Q8400 2.66G (1333MHZ) CPU with 4M: L2 cache, 2GB*1(DDR3 1066) 4*DIMM memory, and 500 GB disk. Machines ran on Windows XP Professional OS. For simplify, we only randomly generated two dimensional coordinates and three dimensional coordinates as two-attribute and three-attribute records for creating 2-d tree and 2D R-tree, respectively, as well as 3-d tree and 3D R-tree, respectively. Both 2D R-tree and 3D R-tree have an associated order (or degree) four. To investigate the scalabilities of *k*-d tree and R-tree, the total numbers of record generated in our databases is varied from 100,000 to 500,000. Our experimentation consists of three parts -- memory cost, time cost for hit data search, and time cost for no hit data search.

### *A. Memory Cost*

In this section, the memories consumed by creating two-attribute and three-attribute indices for two and three dimensional *k*-d trees and R-trees were investigated. The experimental results for two-attribute indices and three-attribute indices are shown in Figure 9 and Figure 10, respectively. From these two figures, we can find that *k*-d trees always outperform R-trees with consuming less memory in both two-attribute and three-attribute indices. This may be due to that R-tree always stores data in leaf nodes and needs to create a number of non-leaf nodes for the minimum bounding rectangles. R-tree also needs to store more coordinates for bounding rectangles and needs more branch links to the child nodes. In contrast, the *k*-d tree likes the binary search tree in which data is stored in either leaf nodes or non-leaf nodes and *k*-d tree has only two branch links to the child nodes. In addition, *k*-d tree need only a two-dimensional or three-dimensional coordinate for each node and does not need to store the boundary for rectangle boxing. In this study, if efficiently using memory is a serious issue for considerations, *k*-d tree will be the better choice.

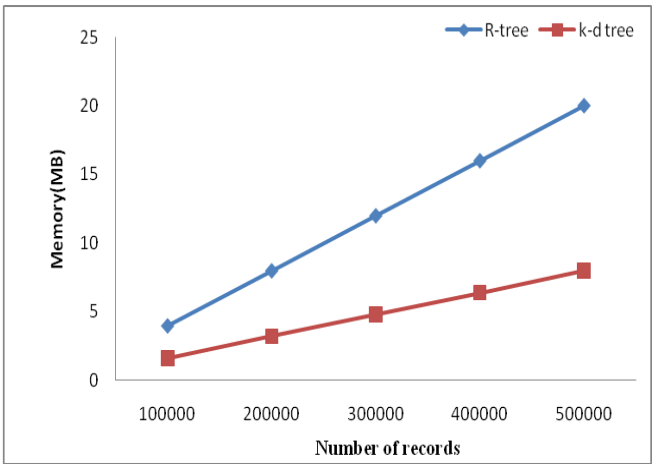


Figure 9. memory cost for two-attribute indices

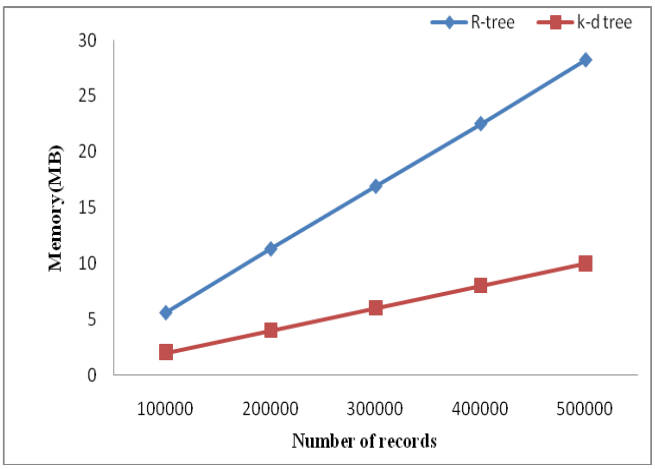


Figure 10. memory cost for three-attribute indices

### *B. Time Cost for Hit Data Search*

In this section, we evaluate the query search efficiency for both *k*-d tree and R-tree. We randomly select 50,000 records from databases to be as the query examples, then search in *k*-d tree and R-tree, which were created in vary database sizes. Then we accumulated the time needed for all query searches. The experimental results are presented in Figure 11 and Figure 12. The Figure 11, demonstrating all the query data searching hit in two-attribute indices, shows that the searching time cost needed in *k*-d tree is far less than in R-tree. Figure 12 which is like in Figure 11 also shows the similar behavior that *k*-d tree outperforms R-tree. The reasons could be that all query examples were selected from databases therefore they were all hit data searches. In *k*-d tree, since either leaf nodes or non-leaf nodes are data nodes, hit data search may benefit *k*-d tree for not necessary always searching to the leaf nodes. However, R-tree only stores data in leaf nodes such that the queries should always search to the leaf node. Although R-tree has the advantage that it uses the minimum bounding

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

boxes to decide whether or not to search inside a sub-tree, this advantage cannot benefit R-tree in this study due to the query searches were all hit.

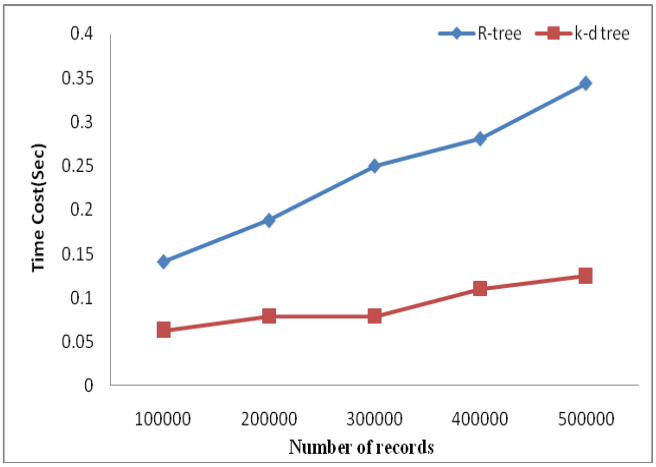


Figure 11. hit data searching in two-attribute indices

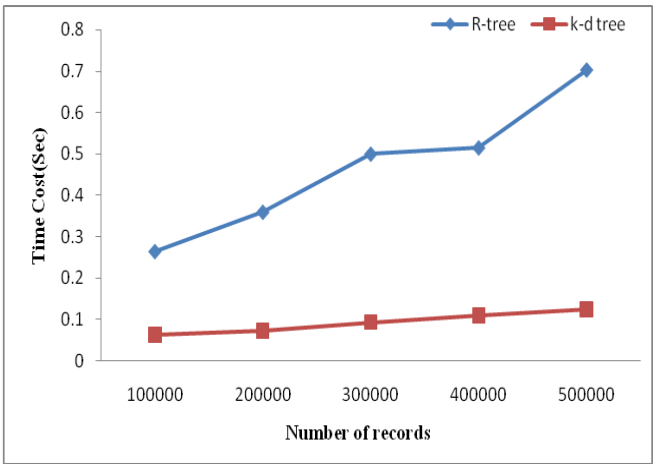


Figure 12. hit data searching in three-attribute indices

### *C. Time Cost for No Hit Data Search*

After we study the time consuming for hit data search, we would like to examine the time cost for no hit data search in this section. As discussed in last section, R-tree has the advantage for using the minimum bounding boxes to decide whether or not to search inside a sub-tree. If a query does not intersect the bounding rectangle, it will be filtered out quickly and not necessary searching down to the leaf nodes. Therefore, no hit data searching might benefit R-tree. In this experiment, we designed and randomly generated 50,000 query samples which cannot be found in our database to insure searching with no hit. These queries are also searched in *k*-d trees and R-trees which represent two-attribute indices and three-attribute indices. Again, we accumulated the time needed for all query searches. Our experimental results are demonstrated in Figure 13 and Figure 14. Obviously, we can find that searching in the *k*-d tree has the lower time cost and still outperforms R-tree in either two-attribute indices or three-attribute indices. It can be explained in the way that searching in *k*-d tree only needs to compare one of the attributes in each node travelled. However, searching in R-tree has to examine the bounding rectangle and to compare every lower bound and upper bound for every attribute (or every dimension) in each node. Therefore, there are more comparisons has to perform in each travelled node of R-tree. Although the curves of R-tree in Figure 13 and Figure 14 are slight lower than in Figure 11 and Figure 12, respectively, the advantage of filtering out no hit query for R-tree is not obvious.

In the research of [16] by Michela and et al. have proved that R-tree is based on minimum bounding rectangles and the three dimensional extension consists of minimum bounding boxes and techniques are often low in efficiency, as sibling nodes might overlap.

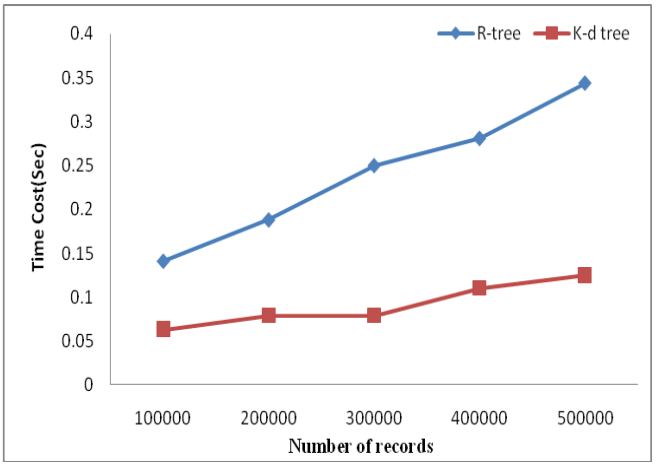


Figure 13. no hit data searching in two-attribute indices

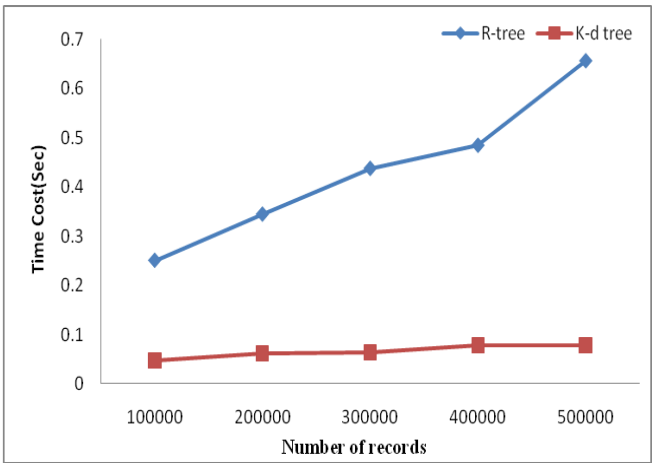


Figure 14. no hit data searching in three-attribute indices

## IV. CONCLUSION

In the Cloud platform, providing scalable database services is an essential requirement. There were few research reports proposed multi-attribute indexing schemes for Cloud platform to manage the huge and variety data to address the complex queries efficiently. These existing and few schemes for Cloud platform were either build local *k*-d tree index for each slave nodes and maintain an R-tree on global master nodes in [23] or use R-trees for both local slave nodes and global master nodes [19]. In this paper, we examined the performances of *k*-d tree and R-tree for two- and three-attribute index structures. Our experimental results are shown that *k*-d tree always outperforms R-tree in either memory consuming and time cost of query searching. It may suggest that applying *k*-d tree for multi-attribute index structure in both local slave nodes and global master nodes for

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Cloud platform would be the better choice due to its efficiency and scalability.

There has an assumption that the data on Cloud platform is distributed into slave nodes by range distribution. Such that a sequence of value intervals of attributes in a slave node can be denoted as a node cube. These node cubes are maintained in the global index of master nodes for pruning irrelevant slave nodes. However, range distributed data may cause load imbalanced in slave nodes due to data may massed in some small range by the property of normal distribution. To address this problem, hash distribution is usually applied but it may lead to poor performance of these proposed index schemes. In the future work, we would like to investigate the load balance issue and also develop a new multi-attribute index structure for Cloud platform to manage the huge and variety data.

## REFERENCES

[1] M.K. Aguilera, W. Golab and M.A. Shah, "A Practical Scalable Distributed B-Tree," in *Proc. of the VLDB Endowment, Vol. 1, Issue 1*, August 2008.

[2] J.L. Bentley, "Multidimensional binary search in database applications", in *IEEE Trans. Software Eng*, Vol. SE-5, Issue 4, pp. 333-340, 1979.

[3] J.L. Bentley, "Multidimensional binary search trees used for associative searching", in *Communications of the ACM* , Vol. 18, Issue 9, pp. 509-517, September 1975.

[4] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: a distributed storage system for structured data," in *proc. of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 15–15, Berkeley, CA, USA, 2006.

[5] B.F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni, "Pnuts: Yahoo!'s hosted data serving platform," in *proc. of Conference on Very Large Data Bases*, pp. 1277-1288, Auckland, New Zealand, August 2008.

[6] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," in *Communications of the ACM , Vol. 51, Issue 1*, January. 2008.

[7] H. Garcia-Molina, J. D. Ullman, and J. Widon, Database System Implementation, Prentice Hall, Inc., Upper Saddle River, NJ, USA, 1999.

[8] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, pages 29–43, October 2003.

[9] C. Gong, J. Liu, Q. Zhang, H. Chen and Z. Gong, "The characteristics of Cloud computing," in *proc. of the 39th International Conference on Parallel Processing Workshops (ICPPW)*, pp.275-279, 2010.

[10] A. Guttman, "R-trees a dynamic index structure for spatial searching," in *proc. of the ACM SIGMOD*, June 1984.

[11] A. Henrich., H. W. Six, and P. Widmayer, "The LSD tree: Spatial access to multidimensional point and non-point objects," in *proc. of Conference on Very Large Data Bases*, pp. 45-53, 1989.

[12] IBM., "Ibm introduces ready-to-use Cloud computing," [Online] Available: http://www-03.ibm.com/press/us/en/pressrelease/22613.ws.

[13] X. L. Liang, J. X. Zhang, H. T. Li and Y. Ping., "Laser radar feature data," *remote sensing information*, 3:71-75, 2005.

[14] M. Lynch, "The Cloud Wars: $100+ billion at stake," *Cloud Computing Expro*, May 2008.

[15] M. Lynch, "Amazon elastic compute Cloud (amazon ec2)," [Online] Available: http://aws.amazon.com/ec2/.

[16] B. Michela, B. schoen, D.F. Laefer and M. Sean "Storage, manipulation, and visualization of LiDAR data," in *proc. of the 3rd ISPRS International Workshop on 3D Virtual Reconstruction and Visualization of Complex Architectures (3D-ARCH)*, Trento, Italy, 25-28 February 2009.

[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *proc. of conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, San Diego, CA, USA, 2001.

[18] L.M. Vaquero, L.R. Merino, J. Caceres, and M. Lindner, "A break in The Clouds: towards a Cloud definition," in *SIGCOMM Computer Communication Review*, Vol. 39, Issue 1, pp. 50-55, January 2009.

[19] J. Wang, S. Wu, H. Gao, J. Z. Li and B. C. Ooi, "Indexing Multi-dimensional Data in a Cloud system", in *proc. of the international conference on Management of data (SIGMOD'10)*, pp.591-602, Indianapolis, Indiana, June 2010

[20] S. Wu and K. L. Wu, "An Indexing Framework for Efficient Retrieval on the Cloud," *IEEE Bulletin of the Technical Committee on Data Engineering*, Vol. 32, No. 1, pp. 75-82, March 2009.

[21] Shufen Zhang, Shuai Zhang, X. Chen and S. Wu, "Analysis and research of Cloud computing system instance," in *proc. of the second international conference on Future Networks*, pp. 88-92, Sanya, Hainan, January 22-24, 2010.

[22] Y. Theodoridis, M. Vazirgiannis and T. Sellis, "Spatio-temporal Indexing for Large Multimedia Applications," in *proc. of the 3rd IEEE ICMCS Conference*, pp.441-448, Hiroshima, Japan, 1996.

[23] X. Zhang, J. Ai, Z. Y. Wang, J. H. Lu and X. F. Meng, "An Efficient Multi-Dimensional Index for Cloud Data Management," in *proc. of the first international workshop on Cloud data management*, pp. 17-24, Hong Kong, November 2009.

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)