SGR-Tree: a Skip Graph based R-Tree for multi-dimensional data indexing in Peer-to-Peer systems

Quang Hieu Vu
ETISALAT BT Innovation Center
Khalifa University of Science, Technology and Research, UAE
quang.vu@kustar.ac.ae

***Abstract*—In this paper, we propose SGR-Tree, an index structure for multi-dimensional data in Peer-to-Peer (P2P) systems. SGR-Tree is an R-Tree index structure constructed on top of a Skip Graph, a P2P overlay network. In SGR-Tree, each Skip Graph node corresponds to an R-Tree leaf node. However, different from R-Tree, SGR-Tree does not employ internal nodes for routing purpose. Instead, at each Skip Graph node, we virtually partition the whole system into non-overlapping regions, each of which is connected to the node via a neighbor node. For each region, the node keeps the minimum hyper-rectangle covering all hyper-rectangles, which are in charged by nodes falling in the region. In this way, when a node issues or receives a query, it simply sends or forwards the query to neighbor nodes whose minimum covering hyper-rectangle intersects with the search region. The main advantage of SGR-Tree is that it can avoid not only the bottleneck problem at the root node but also the high cost of maintaining internal R-Tree nodes, especially when the index structure is often changed. Nevertheless, we prove that SGR-Tree is still able to process multi-dimensional queries efficiently within a boundary of $logN$ steps, where N is the number of nodes in the system. We have done experiments to validate the practicability and efficiency of SGR-Tree.**

Keywords - *Multi-dimensional data indexing, P2P.*

I. INTRODUCTION

In the past decade, Peer-to-Peer (P2P) systems have received a lot of interests from both computer users and researchers. The main advantage of P2P systems is the capability of sharing resources so that large systems can be easily formed by low-cost computers instead of expensive servers. Since sharing data such as images, music files, and textual documents are often represented as multi-dimensional points in a multi-dimensional space, supporting multi-dimensional data indexing in P2P systems is extremely important.

Multi-dimensional data indexing has been well studied in centralized systems. A straightforward method to index multi-dimensional data is to employ Space Filling Curves [1] such as Hilbert curve or Z-curve (Z-order) to first convert multi-dimensional data to one-dimensional data and then to index the converted one-dimensional data in popular one-dimensional index structures. The problem of this method, however, is that it cannot index high-dimensional data efficiently. Alternatively, several index structures that directly index multi-dimensional data such as R-Tree [2], M-Tree [3], and SS-Tree [4] have been proposed. These index structures are generally based on a tree, where the data space is hierarchically divided into smaller subspaces when the space is overloaded and the tree grows up.

A popular approach to support multi-dimensional data indexing in P2P is to adapt centralized multi-dimensional index structures in P2P environment. Given a tree based index structure, the biggest challenge of this approach, however, is how to deal with the bottle-neck problem at the root of the tree structure since all queries need to be started at the root node. VBI-Tree [5] solves this challenge by keeping an upside-path routing table at every node in the tree structure and using upside-paths together with sideways-routing tables for routing purpose. In this way, since a query can start at any node in the tree structure, the bottle neck problem at the root node is eliminated. Nevertheless, the disadvantage of this solution is that it incurs a high cost for updating upside-paths of nodes when the index tree structure is changed.

To avoid the high cost of maintaining upside-paths, we propose SGR-Tree, an R-Tree [2] index structure built on top of a Skip Graph [6]. SGR-Tree uses a different way to build routing tables where no upside-paths are needed. In SGR-Tree, each Skip Graph node corresponds to a leaf node in the R-Tree. For routing purpose, each Skip Graph node virtually partitions the whole system into *non-overlapping* regions, each of which is connected to the node via a neighbor node. For each region, the node keeps the *minimum hyper-rectangle* covering all hyper-rectangles, which are in charged by nodes falling in that region. In this way, a query is still able to start at any node in the Skip Graph structure. When a node issues or receives a query, it needs to send or forward the query to all neighbor nodes whose minimum covering hyper-rectangle intersects with the search region. In addition to SGR-Tree, since load balancing is an important aspect of P2P systems, we also propose a mechanism for load balancing in SGR-Tree. Finally, we conduct experiments to evaluate the performance of SGR-Tree.

The rest of this paper is organized as follows. Section II introduces related work. Section III presents the architecture of SGR-Tree. Section IV describes how query is processed in SGR-Tree. Section V discusses the load balancing mech-

 ISBN: 978-1-61208-173-1

anism. Finally, Section VI shows experimental study and Section VII concludes the paper.

II. Related Work

In general, there are two main approaches to support multi-dimensional data indexing in P2P systems. In the first approach, multi-dimensional data is first converted to one-dimensional data using Space Filling Curves [1]. After that, the result one-dimensional data is indexed to P2P systems supporting one-dimensional data indexing. For example, authors of [7] and [8] share the same idea of using Hilbert curve for data conversion in the first step and Chord [9] for data indexing in the second step. In these methods, the system first encodes each dimensional value to a set of bit keys so that an n-dimensional data item is represented by n sets of bit keys. Hilbert curve is then used to convert these sets of bits keys to a single value for indexing to Chord. On the other hand, authors of [10] use Z-curse for data conversion and Skip Graph [6] for data indexing. The main disadvantage of methods belonging to the first approach is that they are not efficient to index high-dimensional data. Furthermore, these methods often have bad performance when data distribution is skewed since the cost of load balancing is very high.

To overcome the weakness of the first approach, the second approach tries to adapt centralized multi-dimensional data indexing structures in P2P environment. For example, CAN [11], the first P2P system supporting multi-dimensional data indexing, has a structure that is similar to kd-tree [12] and grid file [13]. Alternatively, Skip Index [14] utilizes kd-tree [12] to partition the data space into smaller parts and then maps these parts to Skip Graph [6] by encoding them into unique keys. On the other hand, P2PR-Tree [15] proposes a tree structure, which is adapted from R-Tree [2]. Additionally, VBI-Tree [5] and DP-Tree [16] are designed as frameworks that can deploy different types of index structures such as the R-Tree [2], M-Tree [3], SS-Tree [4] as well as their variants. By employing tree structures in distributed systems, the main challenge of methods belonging to the second approach is how to avoid the potential bottleneck occurred at the root node or nodes near the root since queries always start at the root node. The current solution used by existing index structures is to assign each peer node to represent a leaf node and to let the leaf node keep information about all internal nodes from itself to the root for routing purpose. Since SGR-Tree uses a different way to route queries where information of internal nodes is not needed to maintain, this is the main difference between SGR-Tree and existing index structures.

III. System Architecture

A. Overlay Network

In SGR-Tree, peers participating the system form a Skip Graph structure (i.e., each peer is a Skip Graph node) and each Skip Graph node corresponds to a leaf node in an R-Tree index [1]. While using Skip Graph as the overlay network, SGR-Tree is different from the original Skip Graph structure in three main features.

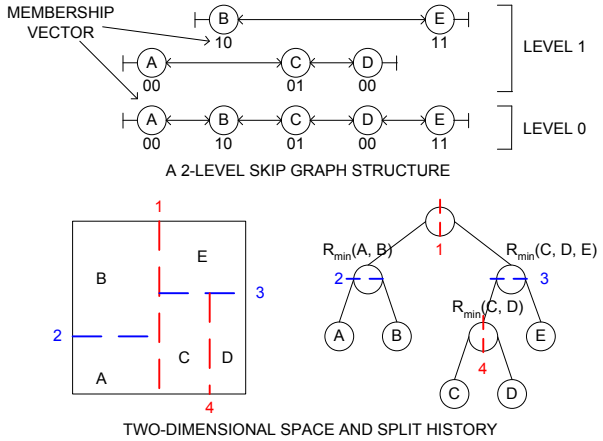


Figure 1. SGR-Tree architecture

- Each node in SGR-Tree is in charge of a *hyper-rectangle* in a multi-dimensional space instead of a range of values as in the original Skip Graph structure.
- To share the load of an existing node for a new coming node, the existing node splits its own hyper-rectangle on *one dimension* into two smaller hyper-rectangles so that the number of data covered by each hyper-rectangle is approximately equal. After splitting, the node on the left of the Skip Graph takes the lower hyper-rectangle while the node on the right takes the upper hyper-rectangle on the split dimension.
- When an existing node leaves the system, it passes its hyper-rectangle to the neighbor node, whose hyper-rectangle shares the border of its hyper-rectangle on one dimension.

An example of an SGR-Tree supporting two-dimensional data indexing with five nodes, A, B, C, D, and E in a two-level Skip Graph is shown in Figure 1 in which the top of the figure displays the Skip Graph overlay network while the bottom of the figure describes hyper-rectangles in charged by each node and the history of splitting the whole two-dimensional space into these hyper-rectangles.

B. Routing Table

SGR-Tree does not use information of internal nodes in the R-Tree structure for routing purpose as existing solutions do. Instead, each node in SGR-Tree creates its own routing table via its neighbor nodes. To create a routing table of a node x, x virtually partitions nodes in the system into *non-overlapping* regions, each of which is connected to x

[1] Since the terms "peer", "Skip Graph node", and "R-Tree leaf node" are interchangeable in our system, we shall simply refer to them as "node" when such reference does not cause any confusion. On the other hand, the term "internal node" in R-Tree shall be kept as it is.

 ISBN: 978-1-61208-173-1

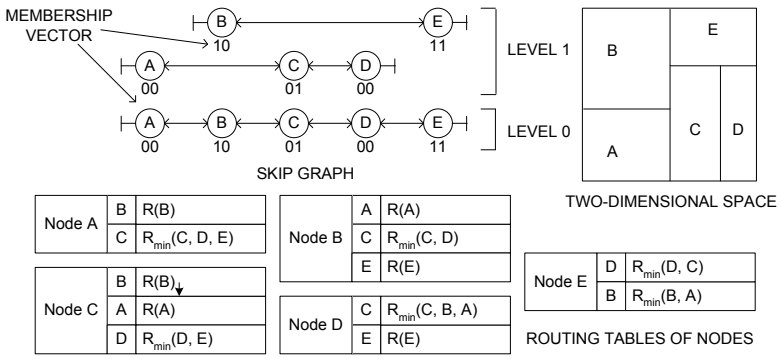


Figure 2. Routing tables of nodes in an SGR-Tree

through a neighbor node. For each region, the routing table of x contains information of the *minimum hyper-rectangle* covering all hyper-rectangles of nodes falling in that region. Given a neighbor y of x at level l in the Skip Graph structure, we define the non-overlapping region connected through y in the view of x as follows.

- If y is a neighbor of x at the highest level, the *non-overlapping region* associated with y consists of all nodes following y on the same side of x, including y. For example, as in Figure 2, since C is a neighbor of A at level 1, which is the highest level in the Skip Graph, the *non-overlapping region* associated with C in the routing table of A consists of C, D, and E (D and E are nodes following C on the right side of A).
- If y is not a neighbor of x at the highest level, the *non-overlapping region* associated with y includes all nodes falling between y and z, the neighbor of x at the nearest level higher than l on the same side of x as y, including y. For example, as in Figure 2, since C is a neighbor of B at level 0, which is not the highest level in the Skip Graph, the *non-overlapping region* associated with C in the routing table of B includes C and D (D is the node falling between C and E, the next neighbor node on the right side of B at level 1).

Figure 2 shows an SGR-Tree index structure with five nodes and details of routing tables at these nodes. In each routing table, the last two columns contain information of neighbor nodes and the *minimum hyper-rectangle* covering all hyper-rectangles of nodes in non-overlapping regions associated with the neighbor nodes. Note that $R(x)$ denotes the hyper-rectangle in charged by x and $R_{min}(x, y, z)$ denotes the minimum hyper-rectangle covering all hyper-rectangles in charged by x, y, and z.

C. Routing Table Construction

To create the routing table of a new node x, each neighbor y of x needs to send to x the *minimum hyper-rectangle* covering the hyper-rectangle in charged by y and all hyper-rectangles in charged by nodes following y on the opposite side with x (this minimum hyper-rectangle can be calculated from the routing table of y). Using received information from neighbor nodes, x builds its routing table as follows.

- If y is a neighbor of x at the highest level in the Skip Graph structure, the *minimum hyper-rectangle* associated with y in the routing table is the *minimum hyper-rectangle* x receives from y.
- If y is a neighbor of x at level l, which is not at the highest level in the Skip Graph structure, the *minimum hyper-rectangle* associated with y in the routing table is the *minimum hyper-rectangle* covering the remainder region of $\mathcal{R}(y) \setminus \mathcal{R}(z)$, where $\mathcal{R}(y)$ is the minimum hyper-rectangle x receives from y and $\mathcal{R}(z)$ is the minimum hyper-rectangle x receives from z, the neighbor of x at the nearest level higher than l on the same side of x as y.

For example, assume that a new node N joins to the right of the existing node B of the SGR-Tree in Figure 2. By joining the system, N takes over a part of the hyper-rectangle in charged by B. The overlay network as well as hyper-rectangles in charged by nodes in the SGR-Tree after the join of N are shown in the top of Figure 3. As the figure shows, the new node N has three neighbor nodes: A at level 1, B at level 0, and C and both levels 0 and 1. Thus, A, B, and C need to send to N the minimum hyper-rectangles covering all nodes following A, B and C in the opposite side of N. They are respectively $R(A)$, $R_{min}(A, B)$, and $R_{min}(C, D, E)$. Since A and C are neighbors of N at the highest level in the Skip Graph structure, $R(A)$ and $R_{min}(C, D, E)$ are also the minimum hyper-rectangles associated with A at C in the routing table of N. On the other hand, since B is not a neighbor at the highest level in the Skip Graph structure, the minimum hyper-rectangle associated with B in the routing table of N is the minimum hyper-rectangle covering $R_{min}(A, B) \setminus R(A) = R(B)$.

Note that since the join of a new node affects the routing tables of the new node's neighbors, neighbors of the new node also need to adjust their routing tables to reflect the existence of the new node and calculate the *minimum hyper-rectangle* associated with the new node from information

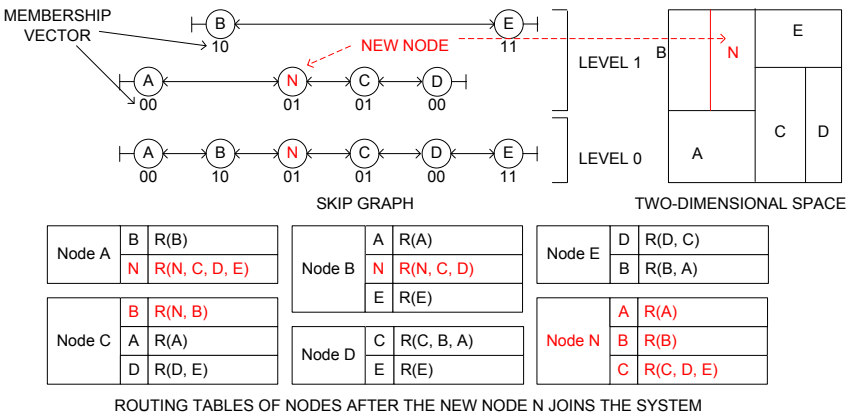


Figure 3. Routing table of the new node N and changes in the routing tables of A, B, and C (neighbors of N) in the SGR-Tree

provided by the new node. Nevertheless, the process of creating the routing table for the new node and updating routing tables of the new node's neighbors incur no additional message cost because the routing information can be piggy-backed with required messages used in the join process (messages used to set up neighbor links of the new node and update neighbor links of existing nodes).

IV. Query Processing

Since a multi-dimensional point query can be considered as a special case of a multi-dimensional range query where the searched region is a point, we only introduce the algorithm for processing a multi-dimensional range query in this section. Basically, when a node x issues a multi-dimensional range query q, x sends q to all neighbor nodes, whose *minimum hyper-rectangle* intersects with the searched region of q. Besides, to avoid sending duplicate query messages to the same node, when x sends q to a neighbor node y at level l, x also determines and sends to y a *lookup region* $\mathcal{L}(y, q)$, which limits neighbor nodes y can further forward q. $\mathcal{L}(y, q)$ is defined as follows.

- If y is a neighbor of x at the highest level in the Skip Graph structure, $\mathcal{L}(y, q)$ are all nodes following y in the opposite side of x.
- If y is not a neighbor of x at the highest level in the Skip Graph structure, $\mathcal{L}(y, q)$ are all nodes falling between y and z, the neighbor of x at the nearest level higher than l on the same side of x as y.

When y receives q from x, if the hyper-rectangle of which y is in charge intersects with the searched region of q, y executes q locally and returns the result to x. Additionally, if there is any neighbor node t of y, which falls into $\mathcal{L}(y, q)$ and has the *minimum hyper-rectangle* intersecting with the searched region of q, y first calculates $\mathcal{L}(t, q)$ based on the position of t and $\mathcal{L}(y, q)$ and then forwards q together with $\mathcal{L}(t, q)$ to t. Note that y does not need to forward q to other neighbor nodes not falling in $\mathcal{L}(y, q)$ because these neighbor nodes should receive the same query q from x or other neighbor nodes of x sooner or later.

For example, assume that node A issues a query q for the shaded region as in Figure 4. While having three neighbor nodes B, C, and D, A only sends q to D since the minimum hyper-rectangle of D, which is $R_{min}(D, E, F, G, H, I, J)$, intersects with the searched region of q (note that $\mathcal{L}(D, q) = \{E, F, G, H, I, J\}$). On the other hand, since the minimum hyper-rectangles of B and C, which are $R(B)$ and $R(C)$, do not intersect with the searched region of q, q is sent to neither B nor C. When D receives q from A, by checking its routing table, D continues to forward the query to E with $\mathcal{L}(E, q) = \{F\}$ and G with $\mathcal{L}(G, q) = \{H, I, J\}$ since E and G are neighbor nodes having minimum hyper-rectangles intersecting with the searched region of q in the opposite side of A. Similarly, q is then continuously forwarded to F with $\mathcal{L}(F, q) = \{\}$ from E; to H with $\mathcal{L}(H, q) = \{I, J\}$ from G; and to I with $\mathcal{L}(I, q) = \{J\}$ from H. Finally, the query is processed locally at F, G, H, and I since hyper-rectangles of which these nodes are in charge intersect with the searched region of q. Note that in Figure 4, routing entries with red and blue colors contain minimum hyper-rectangles that intersects with the searched region of q. However, only neighbor nodes in red routing entries fall in the limited lookup region defined by the query sender, and hence q is only forwarded to these nodes. As an example, when F receives q from E, even though it has two neighbor nodes G and H, whose minimum hyper-rectangle intersects with the searched region of q, it does not forward q to G and H because G and H are not in $\mathcal{L}(F, q)$.

According to the query processing algorithm, when a query q is sent from a node x to a neighbor node y, the lookup region for query processing at y, $\mathcal{L}(y, q)$, is limited by nodes either falling between y and the next neighbor node z at the nearest level higher than the level of y in the same direction with y or all nodes following y if y is the farthest neighbor node in one side of x. This way of limiting the lookup region is actually similar to that of the traditional

 ISBN: 978-1-61208-173-1

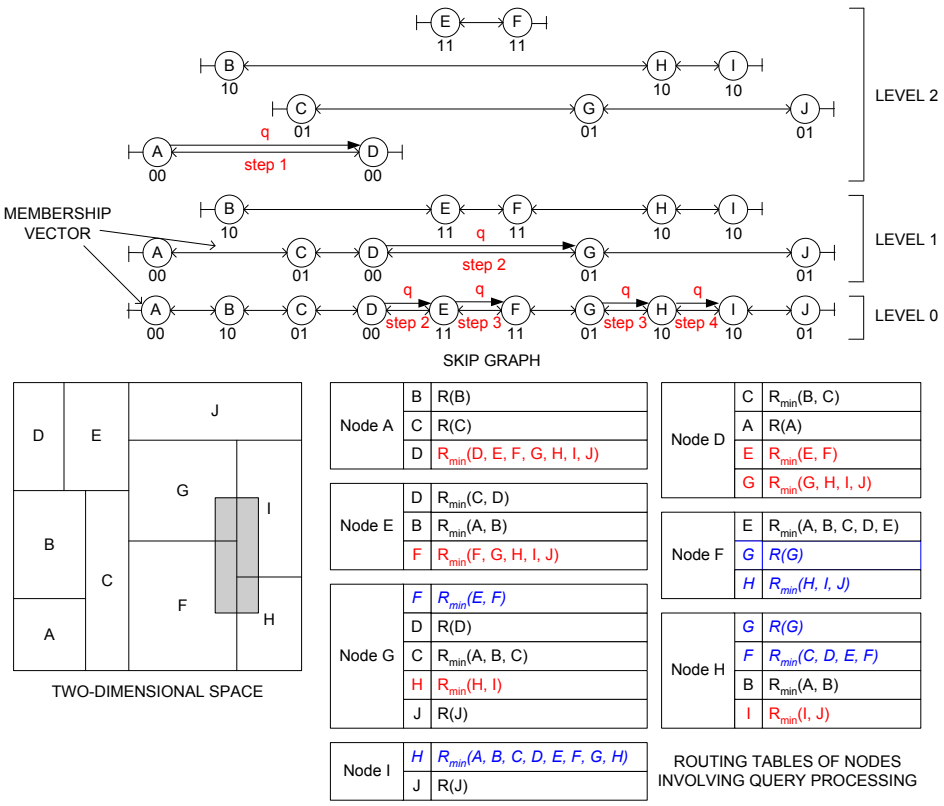


Figure 4. Query Processing in SGR-Tree

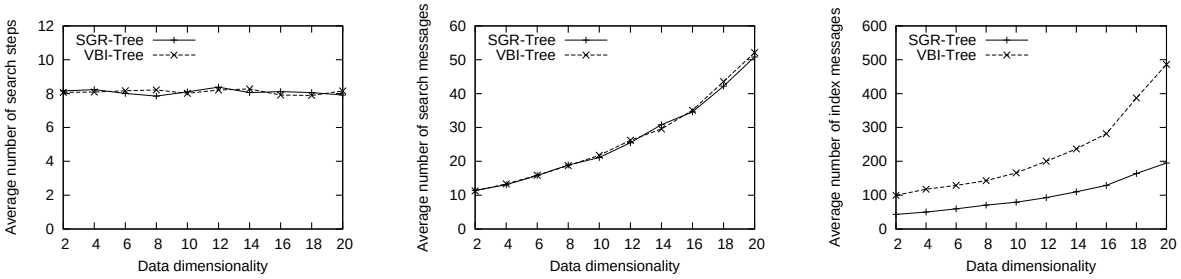
query processing in Skip Graph for single-dimensional range query. The only difference between our query processing algorithm and the traditional query processing is that our algorithm allows to send a query to multiple nodes instead of only one node. As a result, similar to traditional query processing algorithm in Skip Graph, the maximum number of steps required for processing a query in SGR-Tree is also bounded at $O(logN)$.

V. Load Balancing

To serve load balancing purpose, in addition to keeping minimum hyper-rectangles associated with neighbor nodes, we also maintain the number of data belonging to these minimum hyper-rectangles in the routing table. As a result, when a new node joins the system, it can select a heavily loaded node to join as an adjacent to share the heavy load. On the other hand, when a node is overloaded, it can also leverage information in the routing table to search for a lightly loaded node. The lightly loaded node then leaves the system and joins next to the heavily loaded node for load balancing. In other to keep the information up to date, whenever a load of a node is changed by a factor θ, the node sends an update load request to all of its neighbor nodes. When a node receives an update load request from its neighbor, it first updates the load of the minimum hyper-rectangle associated with the sender node. After that, if the new load of the whole group is also changed by a factor θ, the node continues to update its neighbor nodes but the sender node about the change in load.

VI. Experimental Study

To evaluate the performance of SGR-Tree, we implemented a simulator in Java to simulate an SGR-Tree of 10,000 nodes, where we inserted one by one 1,000,000 random multi-dimensional data objects. We tested the simulator with different data dimensionality from 2 to 20 and evaluated the system's performance according to three important criteria: the number of steps required to process a query (search steps), the number of messages required to process a query (search messages), and the number of messages required for building and updating routing tables (index messages). We used VBI-Tree [5] for comparison purpose. The experimental results are shown in Figure 5. The results show that SGR-Tree is comparable to VBI-Tree in terms of search steps and search messages. In particular, in both SGR-Tree and VBI-Tree the number of search steps is independent on data dimensionality while the number of search messages increases with the increasing of data dimensionality. On the other hand, in terms of index messages, SGR-Tree is much better than VBI-Tree. In most cases, SGR-Tree only incurs half of the cost compared to VBI-Tree. This confirms the efficiency of SGR-Tree.



(a) Average number of search steps (b) Average number of search messages (c) Average number of index messages

Figure 5. Performance of SGR-Tree and VBI-Tree

VII. CONCLUSION

In this paper, we have introduced SGR-Tree, a structure that is a combination of Skip Graph and R-Tree to support multi-dimensional data indexing in P2P systems. In SGR-Tree, participant peers form a Skip Graph overlay network in which each Skip Graph node corresponds to a leaf node in the R-Tree structure. For routing purpose, each node builds its own routing table by virtually dividing the whole system into non-overlapping regions, each of which is connected to the node through a neighbor node. For each region, the node maintains the minimum hyper-rectangle covering all hyper-rectangles in charged by nodes in the region. Based on this routing table structure, we have developed a query processing algorithm that can process any multi-dimensional query within $O(logN)$ steps and the query can start at any node in the SGR-Tree. Experiments have been done to evaluate the efficiency of SGR-Tree.

REFERENCES

[1] H. Sagan, *Space-Filling Curves*. Springer-Verlag, 1994.

[2] A. Guttman, "R-trees: a dynamic index structure for spatial searching," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1984, pp. 47–57.

[3] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: an efficient access method for similarity search in metric spaces," in *Proceedings of the 23rd International Conference on Very Large Databases (VLDB)*, 1997, pp. 426–435.

[4] D. A. White and R. Jain, "Similarity indexing with the ss-tree," in *Proceedings of the 12th IEEE International Conference on Data Engineering (ICDE)*, 1996, pp. 516–523.

[5] H. V. Jagadish, B. C. Ooi, Q. H. Vu, R. Zhang, and A. Zhou, "Vbi-tree: a peer-to-peer framework for supporting multi-dimensional indexing schemes," in *Proceedings of the 22nd IEEE International Conference on Data Engineering (ICDE)*, 2006.

[6] J. Aspnes and G. Shah, "Skip graphs," in *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2003, pp. 384–393.

[7] J. Lee, H. Lee, S. Kang, S. Choe, and J. Song, "CISS: an efficient object clustering framework for DHT-based peer-to-peer applications."

[8] C. Schmidt and M. Parashar, "Flexible information discovery in decentralized distributed systems," in *Proceedings of the 12th International Symposium on High Performance Distributed Computing (HPDC)*, 2003.

[9] D. Karger, F. Kaashoek, I. Stoica, R. Morris, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup service for internet applications," in *Proceedings of the ACM SIGCOMM Conference*, 2001, pp. 149–160.

[10] Y. Shu, K.-L. Tan, and A. Zhou, "Adapting the content native space for load balanced," 2004, pp. 122–135.

[11] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proceedings of the ACM SIGCOMM Conference*, 2001, pp. 161–172.

[12] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, no. 9, pp. 509–517, 1975.

[13] K. Hinrichs and J. Nievergelt, "The grid file: a data structure designed to support proximity queries on spatial objects," in *Proceedings of the 9th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, 1983.

[14] C. Zhang, A. Krishnamurthy, and R. Y. Wang, "SkipIndex: towards a scalable Peer-to-Peer index service for high dimensional data," Princeton University, Tech. Rep. TR-703-04, 2004.

[15] A. Mondal, Y. Lifu, and M. Kitsuregawa, "P2PR-Tree: an R-Tree-based spatial index for Peer-to-Peer environments," in *Proceedings of the EDBT Workshop on P2P and Databases (P2PDB)*, 2004.

[16] M. Li, W.-C. Lee, and A. Sivasubramaniam, "DPTree: a balanced tree based indexing framework for peer-to-peer systems," in *Proceedings of the 14th International Conference on Network Protocols (ICNP)*, 2006.

 ISBN: 978-1-61208-173-1