JEAS | Journal of Economics and Administrative Sciences
E-ISSN: 2148-1792 P- ISSN: 1302-2024
Volume 5, Supplement Issue 2
www.jeasweb.org

Implementation and Experimentation of a P2P Network for Dynamic Multidimensional Data Structure[1]

Maryam Moeen Taghavi[*]
Department Computer Engineering, Islamic Azad University South Tehran Branch, Tehran, Iran

Mohammad Ghodsi
Department of Computer Engineering, Sharif University of Technology, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran
***Corresponding author:** Maryam Moeen Taghavi.*

ABSTRACT

Overlay networks and peer-to-peer technologies have become key components for building large scale distributed systems. This paper presents a dynamic multidimensional data structure which we call it skip quadtree for peer-to-peer networks designed for searching in skip list based overlay networks and support range query. This data structure combines the best features of the two well-known data structures, including hierarchical structure search quadtree and skip list. Algorithms for searching, inserting and deleting points in a skip quadtree are fast methods to perform point location and approximate range queries. We focus in this paper on randomized skip quadtree and improved algorithms for searching, inserting and deleting points called improved skip quadtree. The result of this data structure indicates improvement in consumed memory of the network and also it makes less progress in its search algorithm that increases with enlargement of the network.
Keywords: Peer -to-Peer Overlay Networks, Skip quadtree, Improved Skip quad tree.

Introduction

It is extremely important to provide a long list of features such as selection of nearby peers, redundant storage, efficient search/location of data items, data permanence or guarantees, hierarchical naming, trust, authentication and anonymity, P2P overlay networks [1-3]. An effective routing architecture that is self-

[1] **Acknowledgments**

I would like to express my especial thanks of gratitude to my supervisor "Dr. Mohammad Ghodsi" who helped and supported me in completing my thesis.

organizing, massively scalable, and robust in the wide- area, combining fault tolerance, load balancing and explicit notion of locality is encompassed in them.

They embody all the participating peers as network nodes. There are links between any two nodes identifying each other, i.e. if a participating peer knows the location of another peer in a P2P network, then there will be a directed edge from the former node to the latter in the overlay networks. Based on how the nodes in the overlay networks are linked to each other, the classification of the P2P networks are categorized as unstructured, structured and hybrid (multicast).

Structured P2P networks are divided into Distributed Hash Table (DHT) and Non- DHT. DHTs rely on consistent hashing. As a result, they support efficient searches of keys identifying distributed resources while maintaining good load balancing features. Nevertheless, the main pitfall of these systems is that the hashing approach ruins the semantic relation between the keys, and thus, there is no spatial locality among them. Hence, DHTs do not directly support complex queries based on the key ordering, including range queries [4, 5] and nearest neighbor queries [6, 7]. As hashing destroys the logical integrity of the data, such systems cannot support range queries over multi- dimensional data efficiently. Non- DHT networks can solve these problems through their structures [8-11].

Hierarchical subdivisions of space, giving rise to tree- based search structures, typically define linear-space multidimensional data structures. Moreover; a hierarchy is more scalable and it tends to cope better with network dynamics. Skip quadtree is a structured non- DHT P2P network and a hierarchical structure search based on skip lists [12-14]. A skip list is a growing sparse set of sorted doubly linked list of keys, where the higher levels are used as shortcuts to reach nodes at greater distances quickly.

In the paper, it is focused on the randomized skip quadtree called basic skip quadtree, and data structure has been improved in terms of search time, join and leave nodes in P2P overlay networks. Memory consumption and search time requirements of the offered data structure are compared with similar basic skip quadtree that is very efficient. Furthermore, an improved skip quadtree of P2P overlay networks lead to perform more efficiently. The importance of this finding is a high performance data structure in P2P overlay networks to carry out point location and approximate range queries.

Studying hierarchical data structures is of great importance since skip quadtree itself is a kind of hierarchical data structure. Besides; it is described in the skip list, used in this data structure.

The organization of this paper is as follows: Section 2 presents related work, Section 3 describes skip quadtree as randomized, and deterministic, Section 4 considers approximate of range queries, Section 5 discusses about approximate of nearest neighbor queries, and finally section 6 describes some points, which should be considered in P2P networks.

2. Related works

As skip quadtree is a type of hierarchical multidimensional search [15, 16]. Thus, we focus on hierarchical multidimensional search in peer- to-peer networks [17].

2.1. K-d tree [18]

Regions are defined by hyper rectangles in R^d, subdivided into two hyper rectangles using an axis-perpendicular cutting through the median point, for any regions containing more than two points. Therefore, the underlying tree is a balanced binary tree with [long] depth for a static set of points (Figure 1).

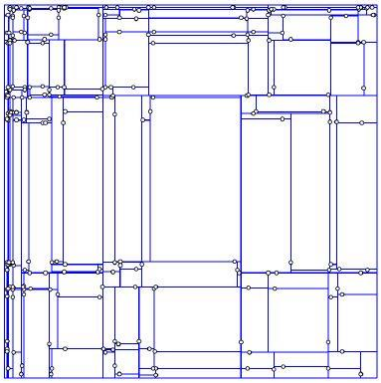

Figure 1. K-d tree Sample.

All cells are rectangles for subdivided splits at median coordinate and they have alternating horizontal and vertical lines, the problems of K- d tree are as follows: 1) High aspect ratio cells 2) No guaranteed query time (too many cells in range) 3) Dynamization is amortized (with approximate median splits).

2.2. Quadtrees

A quadtree is a tree data structure where each internal node has up to four children. Quadtrees are most often used to partition a two dimensional space by recursively subdividing it into four quadrants or regions. The regions may be square or rectangular and they have arbitrary shapes. This data structure was called a quadtree by Raphael Finkel and J.L. Bentley in 1974. A similar partitioning is also known as a Q- tree. All forms of quadtrees share some common traits: 1) they decompose space into adaptable cells 2) Each cell (or bucket) has a maximum capacity. When maximum capacity is reached, the bucket splits 3) The tree directory follows the spatial decomposition of the quadtree (Figure 2).

All cells are squares and once subdivided into splits divided into four equal squares. The problems which quadtree has are as follows 1) Super logarithmic depth 2) Super linear size, 3) No guaranteed query time (too deep recursion) [19].

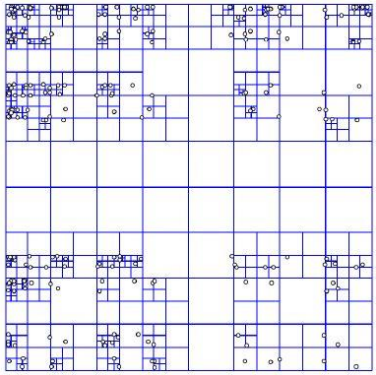

Figure 2. Sample quadtree.

2.3. Compressed quadtree

A compressed quadtree [12-14] is elaborated in terms of an underlying (standard) quadtree for the same point set. Consequently, we define the compressed quadtree by clarifying that squares from the standard quadtree should also be incorporated in the compressed quadtree.

Dimensionality is a significant factor determining regions whether they should be quadtree or octree, but paths in the tree consisting of nodes with only one non- empty child are compressed to a single edge. The compression allows them to still be hypercubes (with optimal aspect ratio), however, it alters the subdivision process from a four- way cut to a reduction to four disjoint hypercubes inside the region. It also drives the height of the (compressed) quad/octree to be at the maximum level of O (n). Certainly, this height bound is still not so efficient.

The definition of a square in a (standard) quadtree is an interesting square if it either is the root of the quadtree or has two or more non- empty quadrants.

The standard quadtree is compressed to explicitly store only the interesting squares, by merging the non- interesting squares and omitting their empty children from the original quadtree, i.e., for each interesting square p, we save 2d bi- directed parent –child pointers, one for each d dimensional quadrant of p. If the quadrant contains two or more points, the pointer will get to the largest interesting square inside that quadrant; when the quadrant contains one point, the pointer get to that point; whenever the quadrant is empty, the pointer is NULL.

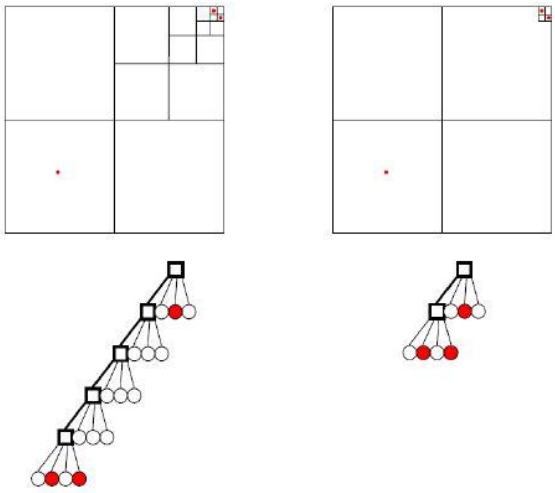

Figure 3. Left: quadtree; Right: compress quadtree.

In Figure 3, it is illustrated that a quadtree containing 3 points located on the left and the corresponding compressed quadtree situated on the right. Two representations of the pointers are depicted in this way: first, a square or an interesting square is drawn as a square, which a quadrant containing a single point is drawn as a solid circle, and latter an empty quadrant is drawn as a hollow circle. The 4- four children of each square or interesting square are ordered from left to right.

THEOREM: Point- location search, point insertion, and deletion in a compressed d- dimensional quadtree of n points can be carried out in O (n) time [13].

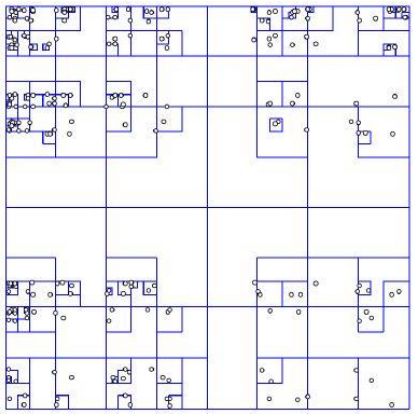

Figure 4. Sample compress quadtree.

3. Skip List

As the bases of many data structures of non-DHT base systems are skip lists [20-23], this section is to clarify them.

A skip list which was invented by William Pugh consists of various layers. The bottom layer is an ordinarily ordered linked list [24]. Each higher layer acts as an "express lane" for the lists below, where an element in layer i emerges in layer i+1 with some fixed probability p which can be 1/2 or 1/4. On average, each element appears in 1/ (1- p) lists, and the tallest element which is usually a special head element at the front of the skip list appears in O ($log^{n}_{1/p}$) lists.

1

1------->4---->6

1---->3->4---->6------->9

1->2->3->4->5->6->7->8->9->10

Figure 5. Sample skip list.

Skip lists are not directly appropriate for use in a distributed environment for several reasons which are as follows:

First, since all operations begin in the highest level of the skip list, which is sparse, these top- level keys become hot- spots and will be involved in an operation with high probability, potentially overwhelming the machines which own them. Furthermore, the sparsity of the top- level list creates single points of failure: if the machines owning these keys go down, the system will be partitioned.

They work well when the elements are inserted in a random order. Skip lists are probabilistic alternatives to balanced trees. Skip lists are balanced by consulting a random number generator. A search for a target element commences with the head element in the top list and goes forward horizontally until the current element is greater than or equal to the target. Whenever the current element is equal to the target, it is found. If the current element is greater than the target, go back to the previous element and drop down vertically to the next lower list, and repeat the procedure.

The expected number of steps in each linked list is easily seen to be 1/p, by tracing the search path backwards from the target to reach an element appearing in the next higher list. Therefore; the total cost of a search is O ($log_{1/p}^{n/p}$), which is O (log n) when p is constant. By choosing different values of p, it is feasible to trade search costs against storage costs.

Given the related works and all features and demerits to get desirable outcome in range query and complex search, we will use non DHT data structure. In this paper, we introduce skip quadtree as non DHT data structure and we will improve its randomized one [25, 26].

4. Skip quadtree

This data structure is originally presented jointly by D. Eppstein, M.T. Goodrich, and J.Z. Sun [12-14]. In skip quadtree a non- negative integer level is allocated to each input point probability 2^{1-i} at each level i. Therefore, for each i, we build a compressed quadtree Q i of points with levels ≤i.

Each interesting square stores seven pointers in this way: next larger interesting square in Qi (if not root), four children (smaller squares), and another one for same square in Q i- 1 (always exists unless i = 0) and another pointer for same square in Q i+1 (if it exists).

There are two definitions of skip quadtree: Randomized and Deterministic.

In this paper, we improved Randomized Skip quadtree.

4.1. Randomized Skip quadtree

The randomized skip quadtree is defined by a sequence of compressed quadtrees respectively defined on a sequence of subsets of the input set S. In particular, we maintain a sequence of subsets of the input points S, such that S0 = S, and, for i > 0, Si is sampled from Si−1 by keeping each point with probability 1/2.

For each Si, we form a unique compressed quadtree Qi for the points in Si. We, therefore, view the Qi's as forming a sequence of levels in the skip quadtree, i.e., S0 is the bottom level (with its compressed quadtree defined for the entire set S) and Stop being the top level, defined as the lowest level with an empty underlying set of points.

Note that if a square p is an interesting square in Qi, then it is also an interesting square in the lower level Qi−1. Indeed, this coherence property among levels in the skip quadtree is what accelerates fast searching.

The sequence of Qi's and Si's, together with these auxiliary pointers define the skip quadtree. (See Figure 6)

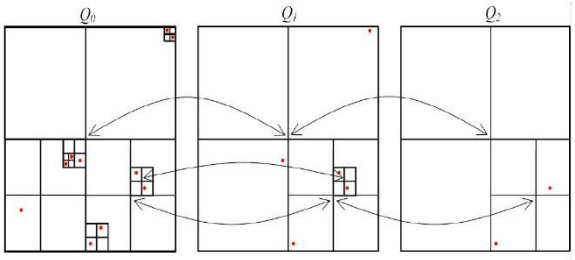

Figure 6. Skip quadtree.

4,1,1. Point Location Search in Randomized Skip quadtree

To find the smallest square in Q0 covering the location of a query point u, we start searching from the root square of the top- level compressed quadtree Ql, here l is the largest value for which Sl is nonempty. Then we search u in Ql as search in compressed quadtree, following the parent- child pointers until we stop at the smallest interesting square pl (u) in Ql that covers the location of u. After we halt searching in each Qi, we go to the copy of pi (u) in Qi−1, denoted by pi- 1, start (u), which equals pi (u), and continue to search u, in Qi−1 starting from this square. This procedure keeps until we reach p0 (u) in the bottom- level compressed quadtree Q0 (See the searching path of x in Figure 7).

If we use Compressed- Search (Qi, p, x) for the subroutine of searching a point location u in a compressed quadtree Qi starting from a square p _ Qi that covers the location of u, then the point location search algorithm in a skip quadtree is as follows:

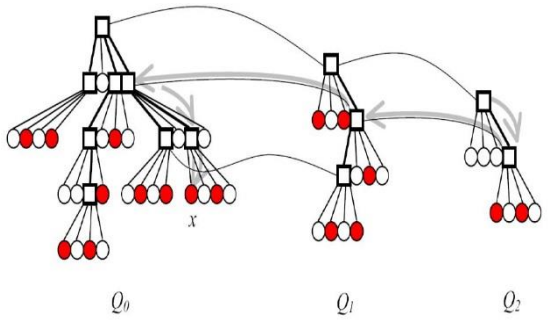

Figure 7. Searching path of x.

Search algorithm for Compressed quadtree:

Pi (u)← Compressed-search (Qi, root (Qi), u)

Require: Qi.

1: for i=1 to n do
2: if (pi (u) =leaf which covering u or pi (u) =internal node with none of its child node, covering u)

{in each i compare with 4 child path}

3: Return pi (u)

4: else

5: u is not in S.

Search algorithm for Randomized Skip quadtree:

Skip-Search (Q, u)

Require: Skip quadtree Q= {Q0,…..,Ql}.

1: Pl (u)← compressed-search (Ql, root (Ql), u)

2: for i= l-1 to 0 do

3: Pi (u) Compressed-Search (Qi, pi+1(u), u)

(Here pi+1(u) =pi, start (u) and pi (u) =pi, end (u).)

4: Return P0(u)

Point Insertion and Deletion in Randomized Skip quadtree

To insert a point x into a randomized skip quadtree Q = {Q0… Ql}, we perform the above point location search, finding pi(x) within all the Qi's, flip coins to find out which Si's x would belong to, then for each Si containing x, we insert x into pi(x) in Qi. Note that by flipping coins, we may create one or more new non- empty subsets Sl+1, . . . , which contain only the point x, and we will consequently create the new compressed quadtrees Ql+1, . . . containing only x and add them into the skip data structure Q.

Deleting a point x is similar to inserting. We search to find pi(x) in all Qi's. Then for each Qi that contains x, delete x from pi(x) in Qi, we may remove Qi from Q if it becomes empty.

As mentioned before, Compressed- Insert (Qi, pi(x),x) and Compressed- Delete(Qi, pi(x),x) the algorithms described in Section 2.3 that insert and delete a point x in a compressed quadtree Qi, respectively, giving the smallest interesting square containing x in Qi are considered as insert and delete algorithm for compressed quadtree shown below.

Insert algorithm for Compressed quadtree:

Compressed-Insert (Qi, pi(x), x)

Require: Qi.

1: Pi (x)← Compressed-search (Qi, root (Qi), x)

2: if pi(x) is empty then insert x with O (1).

3: else

4: if pi(x) have a point y {or an interesting square r) then insert new interesting square q $\subset pi(x)$ {q contains both x and y (or r)}

{4 children creates but 2 of them have data}

5: Return pi(x)

Insert algorithm for Randomized Skip quadtree:

Skip-Insert (Q, x)

Require: Skip quadtree Q = {Q0,……,Ql}.

1:P0(x)← Skip-Search (Q, x)

2: Compressed-Insert (Q0, p0(x), x)

3: for i= 1, 2,…, do

4: if Random (0, 1) variable returns “0” then

5: Return

6: else

7: if i>l then

8: Q ∪ Q {Qi} and l←l+1, where Qi is an empty compressed quadtree.

9: Compressed-Insert (Qi, pi(x), x)

Delete algorithm for Compressed quadtree:

Compressed-Delete (Qi, pi(x), x)

Require: Qi.

1: Pi (x)← Compressed-search (Qi, root (Qi), x)

2: if delete x and pi(x) < 2{no longer interesting} Then delete empty children of pi(x) from Q

3: else

4: delete x

5: Update pi(x)

{If interesting node has 2 children and 1 data (like x) delete then Parent (node) ←child}

5: Return Pi(x)

Delete algorithm for Randomized Skip quadtree:

Skip-Delete (Q, x)

Require: Skip quadtree Q= {Q0,…,Ql}.

1:p0(x)← Skip-search (Q, x)

2: for i=0,…, l do

3: If pi(x) in Qi contains the point x then

4: Compressed-Delete (Qi, pi(x), x)

5: else

6: Return

7: If Qi contains no point then

8: Q←Q\ {Qi,…, Ql}and l←i-l

9: Return

4.2. Deterministic Skip quadtree

In the deterministic version of the skip quadtree data structure, we still keep a sequence of subsets Si of the input points S with S0 = S and build a compressed quadtree Qi for each Si. However, we sample Si from Si−1 in a deterministic way rather than a random sampling.

Our method is to make each Qi and ordered tree so that each Si will be an ordered list, then maintain a deterministic skip list where each Si forms a level to guide the update of the skip quadtree involving of the ordered compressed quadtrees.

We call a compressed quadtree where the quadrants of each square obey the order an ordered compressed quadtree. quadtree puts Q0 as an ordered compressed quadtree for S0 =S. The ordering of the quadrants of each square in Q0 lead to a total order for the points in S, according to the order they appear in tree leaves of Q0 from left to right.

We call the resulting order of data points the order of S0 generated from Q0, and denote it by L0. Make a skip list L for the points in L0 and denote by Li the i- the level of L. Let Si be the subset of S corresponding to Li, i.e., Si contains the data points sampled and promoted to the i- th level Li of the skip list L. Then, we build an ordered compressed

quadtree Qi for each Si.

In lemma [14], it shows that the order generated from each Qi agrees with Li.

Search of a point location in the data structure is done through the skip quadtree part.

Nonetheless, updates consisting insertions and deletions of data points are guided by the skip list part, that is, in building the structure we let the i- th level of L specify the i- th ordered compressed quadtree Qi.

To insert or delete a data point, we first insert or delete it in the skip list L, then update the Qi's according to the changes of L.

5. Improved Skip quadtree

In this paper, we implement Randomized Skip quadtree since this is the basic version of Skip quadtree as you see in the section 4.1 and we call it Basic Skip quadtree. Therefore, we implement two types of P2P overlay networks: Basic Skip quadtree and the other one is Improved Skip quadtree.

Its improved version is called improved skip quadtree. At first, we talk about general structure of basic skip quadtree which we see in simulation [27-32] and then we focus on a general algorithm of improved skip quadtree and after that improvement results about consumed memory and search time between two versions: improved and basic.

5.1. Insert

The difference between insert of improved skip quadtree and basic version is when a data should be inserted in a lower level, instead of creating four children only those that have data should be created. The numbers of child nodes are more than two.

Insert algorithm for Improved Skip quadtree:

Skip-Insert (Q, x)

Require: Skip quadtree Q = {Q0,……,Ql}.

1:P0(x)← Skip-Search (Q, x)

2: Compressed-Insert (Q0, p0(x), x)

3: for i= 1, 2,…, do

4: if Random (0, 1) variable returns "0" then

5: Return

6: else

7: if i>l then

8: Q ∪Q {Qi} and l←l+1, where Qi is an empty compressed quadtree.

9: Compressed-Insert (Qi, pi(x), x)

Insert algorithm for Compressed quadtree:

Compressed-Insert (Qi, pi(x), x)

Require: Qi.

1: Pi (x)← Compressed-search (Qi, root (Qi), x)

2: if pi(x) is empty then insert x with O (1).

3: else

4: if pi(x) have a point y {or an interesting square r) then insert new interesting square q $\subset pi(x)$ {q contains both x and y (or r)}

{for nodes which have data, child creates}

5: Return pi(x)

5.2. Search

In search time, we have just one difference. When the considered data should be searched in the children of an interesting node, it has two children in the best situation. Then, the algorithm compares the child with only two paths not four paths. In this way, it can have three children or in the worst-case four children that is the same as the basic version. In general, its network and search time are developed.

Search algorithm for Improved Skip quadtree:

Skip-Search (Q, u)

Require: Skip quadtree Q= {Q0,…..,Ql}.

1: Pl (u)← compressed-search (Ql, root (Ql), u)

2: for i= l-1 to 0 do

3: Pi (u) Compressed-Search (Qi, pi+1(u), u)

(Here pi+1(u) =pi, start (u) and pi (u) =pi, end (u).)

4: Return P0 (u)

Search algorithm for Compressed quadtree:

Pi (u)← Compressed-search (Qi, root (Qi), u)

Require: Qi.

1: for i=1 to n do

2: if (pi (u) =leaf which covering u or pi (u) =internal node with none of its child node, covering u)

{in each i in best case compare with 2 child path and in worst case with 4 child path}

3: Return pi (u)

4: else

5: u is not in S.

5.3. Delete

It resembles delete in basic skip quadtree but with partial differences in update action. Whole performance of both versions is the same but the only difference in deleting data from interesting is that data and node are deleted together.

Delete algorithm for Improved Skip quadtree:

Skip-Delete (Q, x)

Require: Ski quadtree Q= {Q0,…,Ql}.

1:p0(x)← Skip-search (Q, x)

2: for i=0,…, l do

3: If pi(x) in Qi contains the point x then

4: Compressed-Delete (Qi, pi(x), x)

5: else

6: Return

7: If Qi contains no point then

8: Q←Q\ {Qi,…, Ql}and l←i-l

9: Return

Compressed-Delete (Qi, pi(x), x):

Require: Qi.

1: Pi (x)← Compressed-search (Qi, root (Qi), x)

2: if delete x and pi(x) < 2{no longer interesting} Then delete empty children of pi(x) from Q

3: else

4: delete x

5: Update pi(x)

{if interesting node have 2 child and 1 data (like x) delete then Parent (node) ←child}

{if x delete from interesting node then interesting node must delete}

5: Return Pi(x)

6. Evaluation and Results

In this test, we used consumption memory standard for node and search time. It is clear that the lower the consumption memory is in the network, the higher the performance will be, and the lower the cost will be. The search time is one of the most important parameters considered in a p2p network.

When search time in specific network is low, then this network is useful for query (Figure 8).

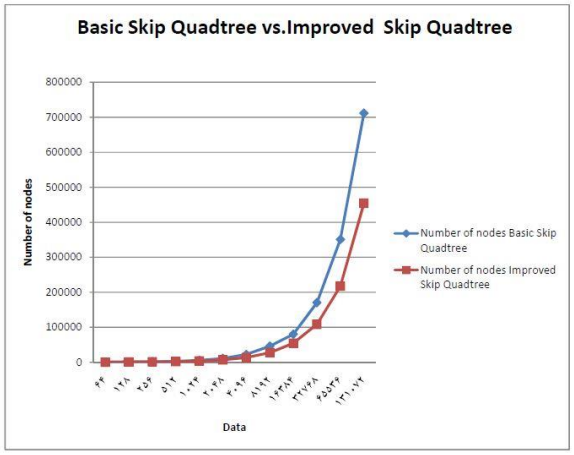

Figure 8. Consumed memory comparisons (part 1).

The horizontal axis is data, which we insert as an input. Range of data is considered from 2^6 to 2^{17}.Vertical axis is the number of nodes. The chart presents consumed memory according to the input data. With increasing data input, number of nodes in Improved Skip quadtree - as you can see in Figure 9 – decreases and performance of the network goes up.

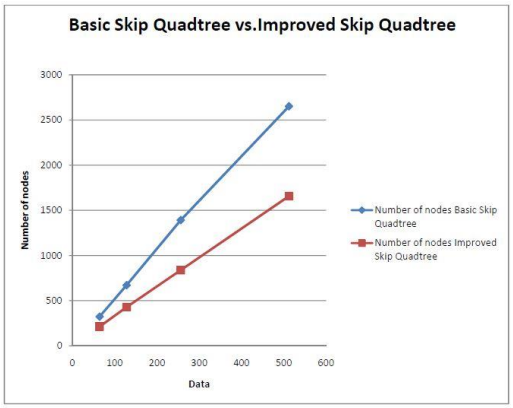

Figure 9. Consumed memory comparisons (part 2).

The horizontal axis is data, which we insert as an input. Range of data is considered between 2^6 to 2^9. This chart is presented because variation does not clear in lower range in the previous chart. Vertical axis is number of nodes, which present consumed memory according to the input data. Number of nodes in Improved Skip quadtree is lower than Number of nodes in Basic Skip quadtree.

Figure 10. Search time comparisons (part 1).

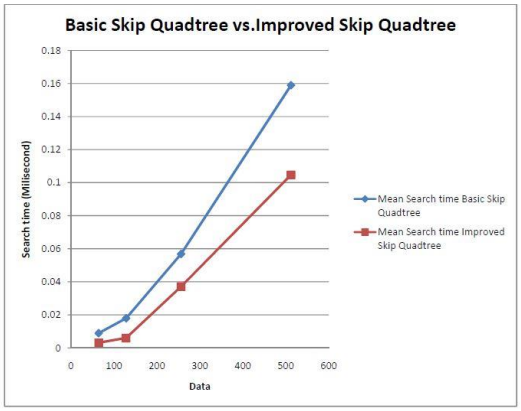

Figure 11. Search time comparisons (part 2).

The horizontal axis is data, which we insert as an input. Range of data is considered from 2^6 to 2^{17}. Vertical axis is search time, which presents mean search time according to the input data. By increasing data input, mean search time in Improved Skip quadtree - as you can see in Figure 10 decreases and performance of network goes up. You can find each item in Improved Skip quadtree faster than Basic Skip quadtree.

In Figure 11 the horizontal axis is data, which we insert as an input. Range of data is considered between 2^6 and 2^9. This chart is presented because variation does not clear in the lower range in previous chart.

Vertical axis is search time, which presents mean search time according to the input data.

Search time in Improved Skip quadtree is faster than search time in Basic Skip quadtree

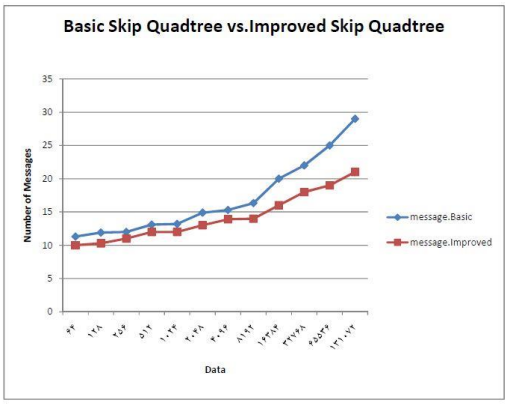

Figure 12. Number of Messages in search time comparisons

The horizontal axis is data, which we insert as an input. Range of data is considered from 2^6 to 2^{17}.Vertical axis is number of messages in search time according to the input data.

With increasing data input, the number of messages in Improved Skip quadtree - as you can see in Figure 12 decreases and performance of network goes up. By observing these charts, it can be concluded that variation of message numbers is the same as search time.

7. Conclusion and Future Directions

In this paper, we presented an improved data structure for routing in P2P networks based. This data structure is multidimensional that in fact is a structured overlay and non-DHT base network.This data structure is improved version of randomized skip quadtree which we improved its efficiency with some changes. This data structure is useful for the range query, which its algorithm has not been implemented for presented version, and this structure can search on multidimensional data and it does not require balancing tree and unique distribution of keys.In general, we presented an improved skip quadtree, which has good efficacy and low cost.Improvements which have been done on this structure include: deleting nodes without data which creates improvement in consumed memory of the network and also creates less improvement in its search algorithm that increases with enlargement of the network.In addition, we compared the improvement rate of node numbers in network and search time with the basic skip quadtree network.

Funding: This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Conflict of interest: The authors declare that they have no conflict of interest.

References

[1] Pujol Ahulló, J., García López, P., Sànchez Artigas, M., Arrufat Arias, M., París Aixalà, M., Bruchmann, M.: PlanetSim: An extensible framework for overlay network and services simulations, Universitat Rovira i Virgili, (2008).
[2] Harvey, N.J.A., Jones, M.B., Saroiu, S., Theimer, M., Wolman, A.: Skip Net: A Scalable Overlay Network with Practical Locality Properties, In: Proceedings of USITS (2003).
[3] Li, M., Chien Lee, W., Sivasubramaniam, A.: Semantic Small World: An Overlay Network for Peer-to-Peer Search, In: Proceedings of the 12th IEEE International Conference on Network Protocols, (2004).
[4] Gonza´lez- Beltra´n, A., Milligan, P.: Range queries over skiptree graphs, Comput. Commun. 1, 358–374 (2008).
[5] Arya, S., Moun, D.M.: Approximate range searching, Comput. Geom. Theory Appl. 17,135-152 (2000).
[6] Arya, S., Mount, D.M.: Approximate nearest neighbor queries in fixed dimensions, In Proc. 4th ACM- SIAM Sympos, Discrete lgorithms (1993).
[7] Arya, S., Moun, D.M., Netanyahu, N.S., Silverman, R., Wu, A.: An optimal algorithm for approximate nearest neighborsearching in fixed dimensions, J. ACM. 45, 891–923 (1998).
[8] Alaei, S., Ghodsi, M., Toossi, M.: Skiptree: A new scalable distributed data structure on multidimensional data supporting range-queries, Comput. Communicat. 33, 73-82 (2010).
[9] Alaei, S., Toossi, M., Ghodsi, M.: SkipTree: A Scalable Range- Queryable Distributed Data Structure for Multidimensional Data, Springer-Verlag Berlin Heidelberg (2005).
[10] Lua, E.K., Crowcrof, J., Pias, M., Sharma, R., Lim, S.: A Survey and Comparison of Peer- to- Peer Overlay NetworkSchemes, In: IEEE Communications Survey, and Tutorial (2004).
[11] Nardelli, E., Barillari, F., Pepe, M.: Distributed Searching of Multidimensional Data A Performance Evaluation Study, J. Parallel Distrib Comput. 49, 111-134 (1998).
[12] Eppstein, D., Goodrich, M.T., Sun, J.Z.: SkipQuadtrees: Dynamic Data Structures for Multidimensional Point Sets, Int. J. Comput. Geom. Ap. 18, 131-160 (2008).
[13] Eppstein, D., Goodrich, M.T., Sun, J.Z.: SkipQuadtrees: Dynamic Data Structures for Multidimensional Point Sets, Int. J. Comput. Geom. Ap. 18, 131-160 (2008).
[14] Eppstein, D., Goodrich, M.T., Sun, J.Z.: Skip Quadtrees: Dynamic Data Structures for Multidimensional, In: SCG'05: Proceedings of the twenty- first annual symposium on Computational geometry (2005).
[15] Tran, D.A., Nguyen, T.: Hierarchical multidimensional search in peer- to- peer networks, Computer Communicat. 1, 346–357 (2008).
[16] Samet, H.: Foundations of Multidimensional and Metric Data Structures, Morgan-Kaufmann publishers (2005).
[17] Stoica, I., Morris R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications, In: Proceedings of the ACM SIGCOMM '01 Conference (2001).
[18] Lia, B., Meng, Q., Holstein, H.: SimilarityK- d tree method forsparse point pattern matching with underlying nonrigidity, Pattern Recognit. 38, 2391-2399 (2005).
[19] Tanin, E., Harwood, A., Samet, H.: Using a Distributed Quadtree Index in Peer-to-Peer Networks, VLDB J. 16, 165–178 (2007).
[20] Pugh, W.: Skip lists a probabilistic alternative to balancedtrees, Commun ACM 33, 540-545 (1990).
[21] Pugh, W.: A Skip List Cookbook, Technical Report CSTR-2286.1, University of Maryland.
(1990).
[22] Munr, J., Papadakis, T., Sedgewick, R.: Deterministic skip lists. In: Proceedings of the third annual ACM- SIAM symposium on Discrete algorithms (SODA) (1992).
[23] Munro, J.I., Papadakis, T., Sedgewick, R.: Deterministic skip lists. In Proc. Annual ACMSIAM Symposium on Discrete Algorithms (SODA) (1992).
[24] Dietz, P.F., Sleator, D.D.: Two algorithms for maintaining order in a list. In Proc. 9th ACM STOC (1987).
[25] Mirrezaei, I., hahparian, J.S., Ghodsi, M.: RAQNet: A Topology- Aware Overlay Network, AIMS'2007, LNCS 4543 by Springer- Verlog (2007).
[26] Nazerzadeh, H., Ghodsi, M.: RAQ: A Range- Queriable Distributed Data Structures, In: 31st Annual Conference on Current Trends in Theory and Practice of Informatics (2005).
[27] PlanetSim Website: http://www.planetsim.net, http://planet.urv.es/planetsim/
[28] Naicken, S., Basu, A., Livingston, B., Odhetbhai, S.: A Survey of Peer-to-Peer Network Simulators, In: Proc. Seventh Annual Postgraduate Symposium (2006).
[29] García López, P., Pairot Gavaldà, C., Mondéjar Andreu, R., Pujol Ahulló, J., Moya, R., Navarro, T.: PlanetSim: A New Overlay Network Simulation Framework, In: Software Engineering and Middleware, 4th International Workshop, SEM 2004, Linz, Austria, September 20-21 (2004).
[30] Baker, M., Lakhoo, R.: Peer-to-Peer Simulators, ACET University of Reading, (2007).
[31] Naicken, S., Livingston, B., Basu, A., Rodhetbhai, S., Wakeman, I., Chalmers, D.: The State of Peer-to-Peer Simulators and Simulations, ACM SIGCOMM Comp. Communicat. Rev. 37, 96-98 (2007).
[32] Naicken, S., Basu, A., Livingston, B., Rodhetbhai, S., Wakeman, I.: Towards Yet Another Peer-to-Peer Simulator, In: Proc. Fourth International Working Conference (2006).