# Spatial Exact Match Query Based on the NA-Tree Approach in P2P Systems

Ye-In Chang, Chen-Chang Wu, and Ching-I Wang

Dept. of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan, Republic of China
changyi@cse.nsysu.edu.tw

**Abstract.** In this paper, we propose to apply an NA-tree in the Chord system to encode spatial region data in the data key part used in the hash function to data search. That is, we combine the NA–tree with the Chord system to solve the overlapping problem which the P2PR–tree can not deal with. From our simulation results, we show that the number of visited peers in our approach is less than that in the P2PR–tree.

**Keywords:** Chord system, exact match query, P2P, searching, spatial data.

## 1 Introduction

Spatial data occurs in several important and diverse applications in P2P systems, for example, P2P virtual cities, GIS, development planning, *etc.* For the problem of answering exact queries for spatial region data in the P2P environment, an R–tree based structure probably is a good choice. Since a peer system is dynamic, the global update characteristics of data insertion/delection in an R–tree can not work well in a P2P system. Moreover, the problem of overlaps in an R–tree results in large number of the disk accesses (which will be considered as large number of messages in P2P systems). Although the P2PR–tree [1]can achieve the goal of the local update for data insertion/deletion, the overlapping phenomenon is still hard to solve.

Recently, for region data access, an NA–tree [2] has been proposed which outperforms R–tree–like data structures. It does not have the problem of overlaps which may occur in an R–tree. On the other hand, the Chord system [3] is a well–known P2P system. Since the Chord system is a hash approach, it is easy to deal with data insertion/delection with only local update. Therefore, in this paper, we propose to apply an NA-tree in the Chord system to encode spatial region data in the data key part used in the hash function to data search. Thus, we combine the NA–tree with the Chord system to solve the overlapping problem which the P2PR–tree can not deal with. From our simulation results, we show that the number of visited peers in our approach is less than that in the P2PR–tree.

The rest of the paper is organized as follows. In Section 2, we introduce the P2PR–tree. In Section 3, we present the proposed *NA–tree approach.* In Section

4, we compare the performance of our approach with the P2PR–tree. Finally, we give a summary.

## 2 P2PR-Tree

Yilifu *et al.* proposed P2PR–tree strategy for object indexing in 2D–space [1] which will have only local update to the proposed index structure when data insertion/delection occurs. In their strategy, a MBR represents the region information that a peer own. In Fig. 1–(a), the MBR $P1$ represents that peer 1 owns the information of this region. As shown in Fig. 1–(b) and Fig. 2), when data $P13$ is inserted, only one path needs to be updated. Therefore, the P2PR–tree does not need global update for data insertion/deletion like the R–tree.

Although the P2PR–tree can achieve the goal of the local update for data insertion/ deletion, the overlapping phenomenon is still hard to solve. Take Fig. 1 as an example. If peer 9 wants to find the spatial region at $P12$, it needs to traverse the P2PR–tree for three branches. Because the region of $P12$ has the overlapping phenomenon, peer 9 has to search the branches which are related to the spatial region data until the spatial region data is found. Here, peer 9 needs to search the first, second, and fourth branches in the R–tree. Therefore, when the overlapping phenomenon occurs very often, it will cost much time to search the data.

## 3 An NA–Tree Approach

In this Section, we present how to answer spatial exact match queries in P2P systems. First, we describes the details of our structure. Next, we present our proposed algorithm for performing insertion operations. Then, we use an example to illustrate the process of the exact match.

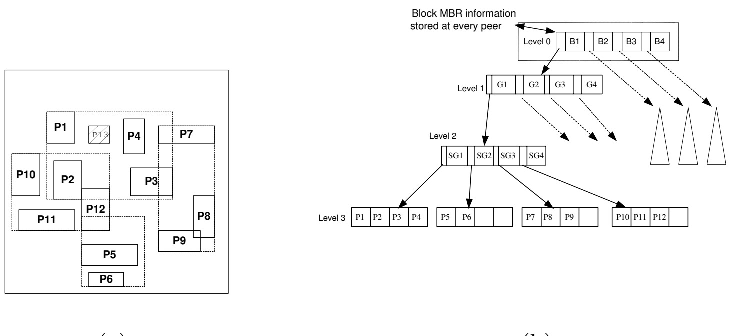


**Fig. 1.** An example: (a) spatial region data; (b) the P2PR-tree

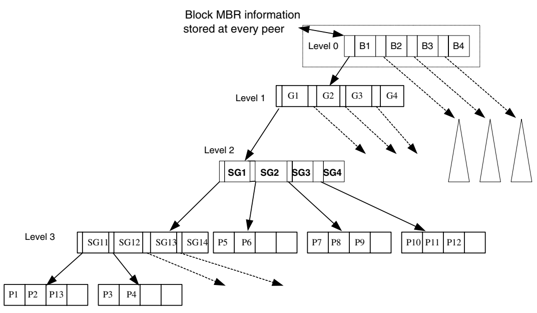


**Fig. 2.** An example of the P2PR–tree of a new addition data

## 3.1 Data Structure

In our method, we apply the NA–tree structure [2] to be as a spatial data index in P2P systems. By using the index of an NA–tree, we can assign an object to a peer in Chord. That is, we use an NA–tree approach in P2P systems. In an NA–tree, an internal node can have nine, four, or two children, and a leaf node is a terminal node. Data can be stored in an internal or a leaf node. An NA–tree is a structure based on data location and organized by the spatial numbers. First, the whole spatial region is decomposed into four regions. We let *regionI* be the bucket numbers between 0 to $\frac{1}{4}$ $(Max\_bucket + 1) - 1$, *regionII* be the bucket numbers between $\frac{1}{4}$ $(Max\_bucket + 1)$ to $\frac{1}{2}(Max\_bucket + 1) - 1$, *regionIII* be the bucket numbers between $\frac{1}{2}$ $(Max\_bucket + 1)$ to $\frac{3}{4}(Max\_bucket + 1) - 1$, and *regionIV* be the bucket numbers between $\frac{3}{4}$ $(Max\_bucket + 1)$ to $Max\_bucket$, as shown in Fig. 3–(a). Based on this decomposition, we find that when an object is lying on the space, only nine cases are possible ( as shown in Fig. 3–(b)).

Nodes in an NA–tree contain index objects entries of the form (*entry_number*, *data*[1..*bucket_capacity*]), where *entry_number* refers to the number of objects in this node, *data*[1..*bucket_bucket*] is an array to store object data, and *bucket_capacity* denotes the maximum number of entries which can be stored in the node.

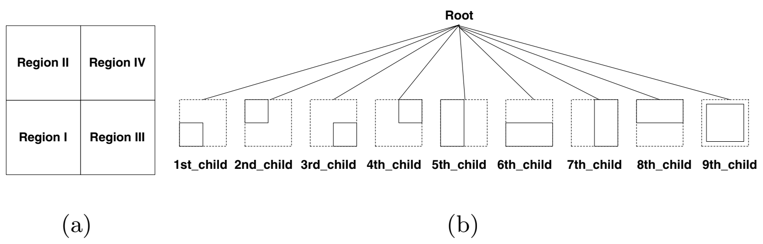


**Fig. 3.** The basic structure of an NA–tree: (a) four regions; (b) nine cases

```
procedure Insertion begin
    l:= The lower left coordinate of the spatial number of an object;
    u:= The upper right coordinate of the spatial number of an object;
    Find the node(nine cases) to which this object belongs at the first level
      according to the spatial number (l, u);
    Calculate the central point of the child node;
    Insert the object into a node in the NA--tree;
    if (node overflows = true)
    begin
      Split the node;
      Dispatch objects into the next level nodes in the NA--tree;
      Re-calculate the central point;
      key := Key_Method_2(p, bp);
    end
    else
      key := Key_Method_1(p, bp);
    Assign the object to the peer or peers in Chord according to the key value;
end;
```

**Fig. 4.** Procedure *Insertion*

## 3.2 The Insertion Algorithm

In this section, we describe our algorithm for inserting spatial data objects into peers in Chord. Procedure *Insertion* is shown in Fig. 4. Basically, we insert a new rectangle into a peer in Chord according the key value which is generated by the NA–tree.

**Insert the Object into a node in the NA-tree.** In procedure *Insertion*, the first step in inserting an object, $O(L, U)$, is to compute its spatial number, *i.e.*, the two bucket numbers of $L$ and $U$. A bucket is numbered as a binary string of 0's and 1's, the so–called $DZ$ expression. The relationship between the space decomposition process and the $DZ$ expression is as follows [4]:

1. Symbols '0' and '1' in a $DZ$ expression correspond to lower and upper half regions, respectively, for each binary division along the $y$–axis. When a space is divided on the $x$–axis, '0' indicates the left half, and '1' indicates right half sub–areas.
2. The leftmost bit corresponds to the first binary division, and the $n$th bit corresponds to the $n$th binary division of the area made by the $(n-1)$th division.

We use two points, $L(X_l, Y_b)$ and $U(X_r, Y_t)$, to record the region of a spatial object. Next, we calculate the corresponding bucket number of $L(X_l, Y_b)$ and $U(X_r, Y_t)$, respectively. Here, we have to convert the bucket numbers from binary to decimal form. The resulting pair of the bucket number is noted as spatial number. That is, we can use the spatial number to record an object. For convenience, we use $O(l, u)$ to denote the spatial number, where $l$ is the bucket number

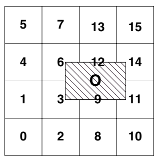
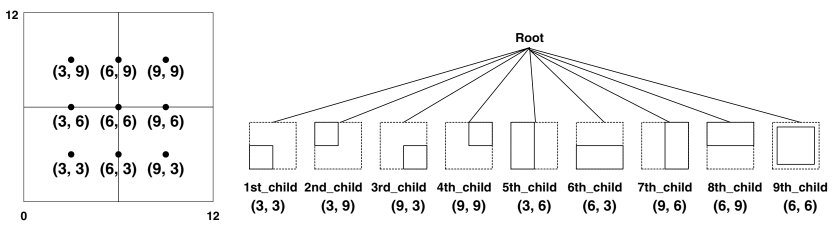


**Fig. 5.** An example of the bucket numbering scheme, O(l, u) = (3, 14)



(a) (b)

**Fig. 6.** An example: (a) central points of nine cases; (b) central points in the NA-tree

of $L(X_l, Y_b)$ and $u$ is the bucket number of $U(X_r, Y_t)$. According to the spatial number $(l, u)$, we will find the node (nine cases) to which this object belongs at the first level. For example, in Fig. 5, the spatial number of the object $O$ is (3, 14). Since $l \in$ Region I and $u \in$ Region IV, this object belongs to 9th_child.

Then, we need to calculate the central point of the region that this object belongs to. That is, our spatial region is decomposed into four regions. Based on this composition, when an object is lying on the space, only nine cases possible. Each case (or region) has its own central point. In other words, each node in the NA–tree can be represented by its central point. Take Fig. 6–(a) as an example. The range of $x$–axis is from 0 to 12. The range of $y$–axis is from 0 to 12. The 6th_region's central point is (6, 3). Hence, the 6th_child in the NA–tree records the cental point (6, 3). The other eight children in the NA–tree can get their own central points in the same way as shown in Fig. 6–(b).

Next, this object is inserted into this node. In our NA–tree, an object is always inserted into the node at the first level. However, when a node overflows, this overflowing node is split. Then, all objects are dispatched into the next level. Thus, an object can be stored in internal or leaf nodes. After inserting the object into an NA–tree, we have to generate the key value to assign this object to an appropriate peer in Chord. We have two methods to generate the key value of an object. Basically, in both methods, first, we decide the first three bits of a data key. Next, we generate the key value of the remaining bits. Finally, we concatenate the first three and the remaining bits to get the key values of objects.

**Function Key_Method_1.** When an object is inserted into the first level of an NA–tree, function *Key_Method_1* that has three steps is called. First, we use three bits to represent eight cases, because the Chord ring can be split into

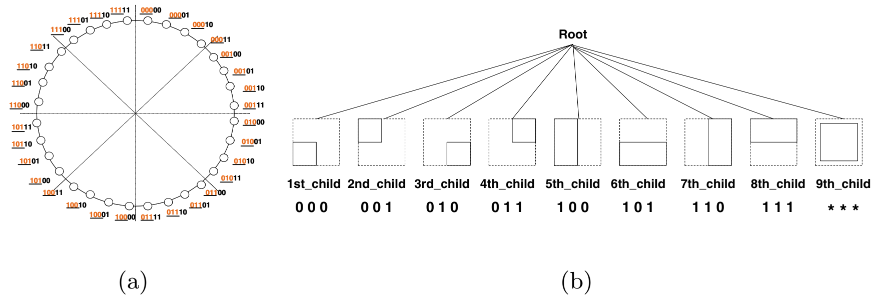


**Fig. 7.** An example: (a) eight partitions in the Chord ring; (b) the bit–expression in each region

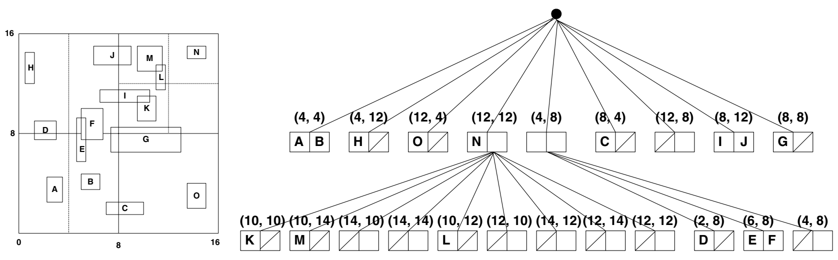
just eight partitions as shown in Fig. 7–(a). Each region in the NA–tree can be expressed by using three bits as shown in Fig. 7–(b). In particular, when the case of the 9th_child occurs, we do not use additional bit–expression to represent it. We still use the eight expression forms described above. We decide to which case the object of 9th_child should belong by the following steps: Each object of 9th_child has its own central point, and the first eight cases (or regions) have their own central point $(C_x, C_y)$. We calculate the distance between the central point of the object and the central points of regions. Then, there will be eight results. The smallest one is our candidate. And, we can decide which case the object belong to according to the shortest distance. If there are more than one candidate, the object belongs to all of them.

Second, we generate the remaining bits by adding $(bp - 3)$ 0's. Finally, we concatenate two bit strings which are calculated by the first two steps. We know that each object must be stored in the first peer of each partition in the Chord ring, because the remaining bits are generated by adding 0's.

**Function Key_Method_2.** To avoid too many data to be stored in the same peer, from the second level of the NA-tree, we call function *Key_Method_2* to calculate the key value of this object. There are three steps in function *Key_Method_2*. First, the first three bits of the key value are inherited from the node's parent. Next, we generate the remaining $(bp-3)$ bits by taking the central point of each region into consideration. Finally, we concatenate the first three and the remaining bits to get the key value of an object.

In this second step that generates the remaining $(bp-3)$ bits, we convert the decimal numbers $(C_x, C_y)$ into binary forms, where $(C_x, C_y)$ is the central point of each region. Next, the binary form of $C_x$ shifts left one bit. Then, we apply the exclusive–OR operation to $C_x$ and $C_y$. We can get a new binary string and choose the last $(bp-3)$ bits to be our remaining bit string.

**Assign the Object to the peer.** When an object is assigned to a peer in Chord, there are two buckets in each peer to store objects. One is to store objects which are owned by the peer now. The other one is to store objects



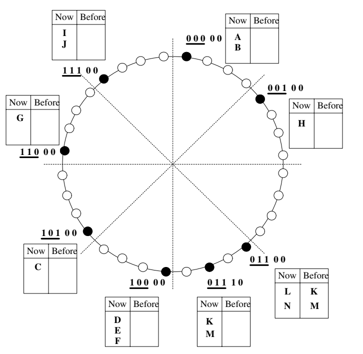
(a) (b)

**Fig. 8.** An example: (a) objects' distribution; (b) the NA–tree structure

which were owned by the peer before. The latter always happens when a node in the NA–tree splits and all objects are re–assigned to peers.

Now, we use one example to describe how the insertion is processed. The spatial distribution of objects is shown in Fig. 8–(a). Objects are inserted into an NA–tree in an alphabetical order. The NA–tree structure is shown in Fig. 8–(b). In our NA–tree, objects can be inserted into the internal or the leaf nodes. Because the capacity of each node is 2, the 4th and the 5th nodes need to be split. Each node has a central point's coordinate. Objects at the first level of the NA–tree use function $Key\_Method\_1$ to calculate their key values. Other objects use function $Key\_Method\_2$.

Let's explain the case of object $M$ in details. Object $M$ belongs to the node whose central point is (10, 14). First, we change the decimal numbers 10 and 14 to binary. We get that the binary forms of 10 and 14 are 1010 and 1110, respectively. Next, the binary form of 10 shifts left one bit resulting in 10100. Then, we will apply the exclusive–OR operation to strings 1010 (10) and 1110 (14). Finally, we can get a new binary sting, 11010, and choose the last two bits, 10, to be our remaining bit string. Finally, a key value of object $M$ is generated by concatenating two binary strings $\underline{011}$ and 10 resulting in $\underline{011}10$. According



| Obj. | Key | |
|---|---|---|
| A | 0 0 0 | 0 0 |
| B | 0 0 0 | 0 0 |
| C | 1 0 1 | 0 0 |
| D | 1 0 0 | 0 0 |
| E | 1 0 0 | 0 0 |
| F | 1 0 0 | 0 0 |
| G | 1 1 0 | 0 0 |
| H | 0 0 1 | 0 0 |
| I | 1 1 1 | 0 0 |
| J | 1 1 1 | 0 0 |
| K | 0 1 1 | 1 0 |
| M | 0 1 1 | 1 0 |
| L | 0 1 1 | 0 0 |
| N | 0 1 1 | 0 0 |

**Fig. 9.** An example: objects in Chord

to each object's key value, each object can be assign to an appropriate peer in Chord. Fig. 9 shows that objects are assigned to peers in Chord according to their own key values.

### 3.3 Answering the Spatial Exact Match Query

When we want to search a spatial object in P2P systems, first, we calculate which region this object belongs to. Then, the key value is generated by function $Key\_Method\_1$ which is described above. By using this key value, we can find a peer in the Chord ring. There are two buckets to store objects in each peer. One is to store objects it has now. The other is to record objects it had before. When we search a peer, there will be three cases:

1. The searching object is in the first bucket.
2. The searching object is in the second one.
3. The searching object is neither in the first bucket nor in the second one.

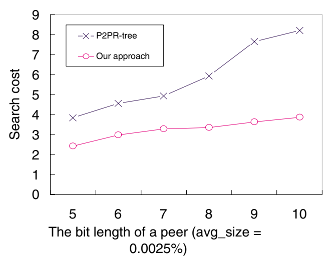
Case 1 means that we find the object and return the result, while Case 3 means that we find nothing and stop searching. In Case 2, it means that the object may be stored in some other peer in Chord. Therefore, we split the region and generate the new key value following function $Key\_Method\_2$. According to the new key value, we search the object again until we cannot find the object in two buckets of the peer.

For example, we want to find object $B$ in Fig. 9. We get that object $B$ belongs to the 1st_child in the NA–tree. Then, we use function $Key\_Method\_1$ to generate the key value, $\underline{000}00$. According to this key value, we search the peer, $\underline{000}00$, in Chord. We can find object $B$ in the first bucket of this peer.

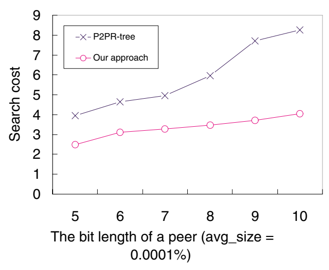
If we want to find object $M$ in Fig. 9. We get that object $M$ belongs to the 4th_child (*i.e.*, $\underline{011}$) in the NA–tree. Then, we use function $Key\_Method\_1$ to generate the key value, $\underline{011}00$. According to this key value, we search the peer, $\underline{011}00$, in Chord. But we can not find object $M$ in the first bucket of this peer. However, we find object $M$ in the second bucket of this peer. Therefore, we split the region 4 and re–calculate the key value by function $Key\_Method\_2$. Because the object belongs to the 4th region, the first three bits are $\underline{011}$. Further, the central point of the region is (10, 14). A new key value, $\underline{011}10$, is generated. According to this key value, we search the peer, $\underline{011}10$, in Chord. We can find object $M$ in this peer's bucket one.

## 4 Simulation Results

In this section, we compare our approach with the P2PR–tree. Here, we define that the search cost in P2P systems is the number of visited peers [5]. Given that the data space is 1000*1000, we took measurements for six different values of the parameter $P$ equal to 5, 6, 7, 8, 9, and 10, respectively. *That is*, there are $2^5$, $2^6$, $2^7$, $2^8$, $2^9$, and $2^{10}$ peers in our measurements. The data objects with the average sizes 0.0025% and 0.0001% are uniformly distributed (without overlap)



(a)



(b)

**Fig. 10.** A comparison of the search cost for processing a exact match query: (a) $avg\_size = 0.0025\%$; (b) $avg\_size = 0.0001\%$

on the whole data space. *Bucket_capacity* was assigned to be 10. For each spatial data file, we create 100 rectangles randomly to do exact match queries, and then calculate the average search cost of them.

Figure 10 shows the average search cost (in terms of the number of visited peers) of our approach and the P2PR–tree. In the P2PR–tree, a MBR is a region data as well as a peer. Hence, when the number of peers increases, the number of objects increases and causes the overlapping problem. As the number of peers increases, the search cost increases. In our approach, a MBR represents an object in the space and peers are distributed in Chord. When the number of peers increases, the objects are assigned to other peers in Chord. Therefore, as the number of peers increases, the search cost increases. From Fig. 10 we observe that our approach needs lower search cost than the P2PR–tree.

## 5 Conclusion

In this paper, based on the NA–tree, we have presented an approach to deal with the spatial region data in the Chord system. The Chord ring is divided into eight partitions. We use three bits to represent it. For remaining bits of a key value, we have proposed two methods to generate it by adding 0's in method 1 or taking the central point of each region into consideration in method 2. The first method is simple and applicable to the case that there are few objects in the P2P system. The second method is applicable to the case that there are too many objects in the P2P system. Then, we can get the key value by concatenating these two bit strings. According to this key value, we can assign data objects to peers in Chord. Our approach can support exact match queries in $2D$ space and reduce the overlapping problem. From our simulation results, we have shown that the number of visited peers in our approach is less than that in the P2PR–tree. Hence, our approach by using the NA–tree in the Chord system has lower search cost than the P2PR–tree.

**Acknowledgments.** This research was supported in part by the National Science Council of Republic of China under Grant No. NSC-97-2221-E-110-058-MY3 and National Sun Yat-Sen University. The authors also like to thank "Aim for Top University Plan" project of NSYSU and Ministry of Education, Taiwan, for partially supporting the research.

## References

1. Mondal, A., Lifu, Y., Kitsuregawa, M.: P2PR–tree: An R–Tree–Based Spatial Index for Peer–to–Peer Environments. In: Proc. of Int. Workshop on Peer–to–Peer Computing and Databases, Crete, Greece, pp. 516–525 (2004)
2. Chang, Y.I., Liao, C.H., Chen, H.L.: NA-Trees: A Dynamic Index for Spatial Data. Journal of Information Science and Engineering 19, 103–139 (2003)
3. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A Scalable Peer–to–Peer Lookup Service for Internet Applications. In: Proc. of ACM SIGCOMM Conf., pp. 149–160 (2001)
4. Ohsawa, Y., Sakauchi, M.: A New Tree Type Structure with Homogeneous Nodes Suitable for a Very Large Spatial Database. In: Proc. of IEEE Int. Conf. on Data Eng., pp. 296–303 (1990)
5. Kwon, O., Moon, J.W., Li, K.J.: DisTIN – A Distributed Spatial Index for P2P Environment. In: Proc. of Data Engineering Workshop, pp. 11–17 (2006)