Spatial data occur in several important and diverse applications in P2P systems, for example, P2P virtual cities, GIS, development planning, etc. 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

In recent years, the research on Peer-to-Peer (P2P) systems has grown significantly. P2P systems can be used for many applications, such as the distributed computing (e.g. SETI@home [4]), file sharing [7] (e.g. Napster [17], Gnutella [1], Freenet [9], FreeHaven [10]), collaboration (e.g. Groove [2]) and platforms (e.g. JXTA [3]) etc. A peer is a node on a P2P network that forms the fundamental processing unit of any P2P solution. A P2P system is constructed by a large, dynamic set of peers which are distributed over a wide-area network. P2P overlay networks are distributed systems in nature, without any hierarchical organization or centralized control. As with any computing system, the goal of P2P systems is to support applications that satisfy the needs of users. Selecting a P2P approach is often driven by the following goals: cost sharing and reduction, improved scalability and reliability, resource aggregation, increased autonomy, anonymity and privacy and enabling ad-hoc communication [31].

Spatial data occur 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/deletion 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 [13] 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 [8] 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 [15] is a well-known P2P system. Since the Chord system is a hash approach, it is easy to deal with data insertion/deletion 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. 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 Peer-to-Peer systems and exact match query in P2P systems. 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 Related Work

In this section, we first describe Peer-to-Peer systems. Then, we give a survey of the exact match query in P2P systems.

2.1 Peer-to-Peer Systems

A taxonomy of computer systems from the P2P perspective is shown in Figure 1 [31]. All computer systems can be classified into centralized and distributed systems. Centralized systems are represented by mainframes and supercomputers for execution of compute-intensive applications and large batch jobs, and high-end servers for data or content sharing and for collaborative applications. Distributed systems are those in
which components located at network computers communicate and coordinate their actions only by passing messages [5].

Distributed systems can be further classified into the client-server model and the P2P model. A client is informally defined as an entity (e.g. node, program, module etc.) that initiates requests but is not able to serve requests. If the client also serves the request, it plays the role of a server. A server is informally defined as an entity that serves requests from other entities, but does not initiate requests. If the server does initiate requests, then it plays the role of a client. Typically, there are one or a few servers versus many clients. A client-server model represents the execution of entities with the roles of clients and servers. Any entity in a system can play both roles but for a different purpose, i.e. server and client functionality residing on separate nodes.

Similarly, an entity can be a server for one kind of request and client for others. The P2P model enables peers to share their resources (information, processing, presence etc.) with at most a limited interaction with a centralized server. The peers may have to handle a limited connectivity (wireless, unreliable modem links etc.), support possibly independent naming, and be able to share the role of the server. It is equivalent to having all entities being client and servers for the same purpose.

By considering whether the overlay network is created as nodes and content are added, or its creation is based on specific rules. The P2P model can be classified into unstructure and structure by the network structure [5]. Unstructure means that the placement of content (file) is completely unrelated to the overlay topology. Structure means that the overlay topology is tightly controlled and files are placed at precisely specified locations. In the unstructured network, it is classified into the purely, hybrid decentralized model and the partially centralized model.

2.2 Exact Match Query in P2P systems

For the problem of answering exact queries for spatial data in P2P environments, Ganesan et al. proposed a method called SCRAP [11]. They mapped multi-dimensional data down to a single-dimension using z-ordering [6]. Then, this data is range-partitioned across peers. Schmidt et al. proposed a method, using space filling curve to assign data to peers [16]. Both of them solve the problem of point data. However, they cannot deal with the spatial region data.

Basiclly, for the problem of answering exact match queries for the spatial region data, there are three different types of traditional methods. The first method is based on the order of the spatial region data, i.e., space filling curve. The second method is by constructing the spatial space, i.e., quadtree. The third method is by constructing the distribution of the spatial region data, i.e., R-tree. To answering the exact match queries for the spatial region data, the first and second methods have some disadvantages.

For the space filling curve, because the numbers for the spatial region data are not continuous, it needs to access the disk for more than one time. For the quadtree, because the spatial region data may need to store in more than one node, it needs to traverse these nodes which have duplicated data. Both of them cause a large number of disk accesses. However, the R-tree uses the MBR to organize these spatial region data objects based on their distribution, which could reduce the number of disk accesses. Therefore, the R-tree probably is a good choice.

Yilifu et al. proposed the P2PR-tree strategy for object indexing in 2D-space [13] which will have only local update to the proposed index structure when data insertion/deletion occurs. In their strategy, a MBR represents the region information that a peer owns. In Figure 2-(a), the MBR $P1$ represents that peer 1 owns the information of this region. As shown in Figure 2-(b) and Figure 2-(c), 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 Figure 2-(a) 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 describe 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.

3.1 Data Structure

In our method, we apply the NA-tree structure [8] 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, we can assign an object to a peer in Chord. Then, we use an example to illustrate the process of the exact match.

1. **1st_child**: If both \( l \) and \( u \) ∈ region I, \( O \) is the first child of node \( p \).
2. **2nd_child**: If both \( l \) and \( u \) ∈ region II, \( O \) is the second child of node \( p \).
3. **3rd_child**: If both \( l \) and \( u \) ∈ region III, \( O \) is the third child of node \( p \).
4. **4th_child**: If both \( l \) and \( u \) ∈ region IV, \( O \) is the fourth child of node \( p \).
5. **5th_child**: If \( l \) ∈ region I and \( u \) ∈ region II, \( O \) is the fifth child of node \( p \).
6. **6th_child**: If \( l \) ∈ region I and \( u \) ∈ region III, \( O \) is the sixth child of node \( p \).
7. **7th_child**: If \( l \) ∈ region III and \( u \) ∈ region IV, \( O \) is the seventh child of node \( p \).
8. **8th_child**: If \( l \) ∈ region II and \( u \) ∈ region IV, \( O \) is the eighth child of node \( p \).
9. **9th_child**: If \( l \) ∈ region I and \( u \) ∈ region IV, \( O \) is the ninth child of node \( p \).

In the above nine children, there are three kinds of data structure. The data structure of **1st_child**, **2nd_child**, **3rd_child**, **4th_child** is shown as follow:

```
struct nine_children{
    struct nine_children *1st_child;
    struct nine_children *2nd_child;
    struct nine_children *3rd_child;
    struct nine_children *4th_child;
    struct three_children *5th_child;
    struct three_children *6th_child;
    struct three_children *7th_child;
    struct three_children *8th_child;
    struct one_list_children *9th_child;
}
```

The data structure of **5th_child** and **7th_child** is shown as follow:

```
struct three_children{
    struct three_children *5th_child;
    struct three_children *7th_child;
    struct one_list *9th_child;
}
```

The data structure of **6th_child** and **8th_child** is shown as follow:

```
struct three_children{
    struct three_children *6th_child;
    struct three_children *8th_child;
    struct one_list *9th_child;
}
```

The data structure of **9th_child** is as follow:

```
struct one_list{
    data_object[1..bucket_capacity];
    struct one_list *next_ptr;
}
```

Nodes in an NA-tree contain index objects entries of the form (**entry_number**, **data[1..bucket_capacity]**), where
In this section, we describe our algorithm for inserting spatial data objects into peers in Chord. Procedure Insertion is shown in Figure 5. Basically, we insert a new rectangle into a peer in Chord according the key value which is generated by the NA-tree.

3.2.1 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 [14]:

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 nth bit corresponds to the nth binary division of the area made by the \((n-1)\)th division.

![Figure 6: An example of the bucket numbering scheme, \(O(l, u) = (3, 14)\).](image)

We use two points, \(L(X_r, Y_r)\) and \(U(X_l, Y_l)\), to record the region of a spatial object. Next, we calculate the corresponding bucket number of \(L(X_r, Y_r)\) and \(U(X_l, Y_l)\), 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 of \(L(X_r, Y_r)\) and \(u\) is the bucket number of \(U(X_l, Y_l)\). 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 Figure 6, the spatial number of the object \(O\) is \((3, 14)\). Since \(l\) is contained in \(Region\; I\) and \(u\) is contained in \(Region\; IV\), this object belongs to the 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 are 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 Figure 7-(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 central point \((6, 3)\). The other eight children in the NA-tree can get their own central points in the same way as shown in Figure 7-(b).

Next, this object is inserted into this node. In our NA-tree, an object is always inserted into the node at the

### Procedure Insertion

```plaintext
procedure Insertion
    \(l\) := 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;
```

![Figure 5: Procedure Insertion.](image)
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. When a split occurs, each node will create a different number of children, depending on different cases of $parentid$ and $uid$, where $parentid$ and $uid$ are used to record the type of its parent and itself, respectively, $1 \leq i \leq 9$. When the splitting node's $uid$ is 1, 2, 3, or 4, it will create nine child nodes. If the splitting node's $uid$ is 5 or 7, it will create three child nodes, i.e. $5th\_child$, $7th\_child$, and $9th\_child$. If the splitting node's $uid$ is 6 or 8, it will create three child nodes, i.e. $6th\_child$, $8th\_child$, and $9th\_child$. In particular, when $uid$ is 9, the location of the centroid of the rectangle is used to decide which region it should belong to [8].

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.

### 3.2.2 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. This function requires two parameters $p$ and $bp$, where $p$ is a node in the NA-tree and $bp$ is the size of a data key.

First, we use three bits to represent eight cases, because the Chord ring can be split into just eight partitions as shown in Figure 8-(a). Each region in the NA-tree can be expressed by using three bits as shown in Figure 8-(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 belongs 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.

### 3.2.2 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 $C_y$ 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.

### 3.2.3 Assign the Object to the peer

When an object is inserted into the first level of an NA-tree, we use $Key\_Method\_1$ to generate the key value. Because when the number of objects is very small, objects are
always inserted into the node at the first level in the NA-tree. 

Key_Method_1 is simple and fast to generate the key values. But, when there are many objects in the NA-tree, we can not use Key_Method_1 only since it will cause too many objects to be assigned to the same peer in Chord. There will be heavy overload in this peer. Therefore, from the second level of the NA-tree, when objects are inserted, we use Key_Method_2 to generate the key values. Then, the overload of this peer will be distributed to other peers. According to the key value which is generated by function Key_Method_1 or function Key_Method_2, the object can be assigned to an appropriate peer in Chord.

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

Now, we use one example to describe how the insertion is processed. The spatial distribution of objects is shown in Figure 9-(a). Objects are inserted into an NA-tree in an alphabetical order. The NA-tree structure is shown in Figure 9-(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 of the first level use function Key_Method_1 to calculate their key values. Other objects use function Key_Method_2.

We take objects belong to 4th_child in the NA-tree as an example. Because object N is in the first level, the key is generated by function Key_Method_1. Then, we can get the key value 01100. Because objects K, M, L are in the second level, the key value is generated by function Key_Method_2. Since these three objects K, M, L belong to the 4th_child, we can get the first three bits are 011. The remaining two bits are generated by their central points.

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 011 and 10 resulting in 01110. According to each object's key value, each object can be assign to an appropriate peer in Chord. Figure 10 shows that objects are assigned to peers in Chord according to their own key values.

3.2 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 Figure 10. 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, 00000. According to this key value, we search the
peer, 0000, in Chord. We can find object B in the first bucket of this peer.

If we want to find object M in Figure 10, we get that object M belongs to the 4th child (i.e., 011) in the NA-tree. Then, we use function Key_Method_1 to generate the key value, 01100. According to this key value, we search the peer, 01100, 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 recalculate the key value by function Key_Method_2. Because the object belongs to the 4th region, the first three bits are 011. Further, the central point of the region is (10, 14). A new key value, 01110, is generated. According to this key value, we search the peer, 01110, in Chord. We can find object M in this peer’s bucket one.

4 Experimental Results

In this section, we compare the performance of our approach with the P2PR-tree. Here, we define that the search cost in P2P systems is the number of visited peers [12]. The parameter $P$ is the bit length of a peer in Chord. That, there are $2^P$ peers in Chord. All data objects which are used in this performance evaluation are randomly generated sets of rectangles. Each rectangle is displaced at random within the given two-dimensional data space; that is, the data are under the uniform distribution. The average size of data objects is $\text{avg}_\text{size}$ which is measured in the percent of the size of the data space.

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) on the whole data space, respectively. The maximum number of data containable in a node 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 11 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 Figure 11 we observe that our approach needs lower search cost than the P2PR-tree.

![Figure 11: A comparison of the search cost for processing an exact match query: (a) $\text{avg}_\text{size} = 0.0025\%$; (b) $\text{avg}_\text{size} = 0.0001\%$.](image)

Next, we compare our approach with the P2PR-tree based on five cases of data distribution shown in Figure 12.

![Figure 12: Five cases of data distribution: (a) uniform; (b) centralization; (c) diagonal; (d) X-parallel; (e) sine.](image)
For each case, we took measurements for six different spatial data files containing 500, 1000, 1500, 2000, 2500, and 3000 rectangles of average size 0.0025%. Bucket capacity was assigned to be 10. Figure 12 shows these five cases of data distribution which are described as follows [8]:

(a) **Uniform Distribution**: The data objects are uniformly distributed on the whole data space.
(b) **Centralization Distribution**: Most of the data objects are centralized in a small region.
(c) **Diagonal Distribution**: The data objects follow a uniform distribution on the main diagonal.
(d) **X-parallel Distribution**: Most of the data objects are located on a line which is paralleled the X-axis.
(e) **Sine Distribution**: The data objects follow a sine curve.

Figures 13, 14, 15, 16, 17 show the comparison with the P2PR-tree when the data files are of the uniform, centralization, diagonal, X-parallel and sine distribution, respectively. As the number of objects increases, the overlapping phenomenon often occurs if objects are centralized. In the P2PR-tree, the search cost increases, because it needs to search peers in different branches. From Figures 13, 14, 15, 16, 17, we observe that our approach has lower search cost than the P2PR-tree in each of six different data distributions.

### 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.

---

![Figure 13](image1.png)

**Figure 13**: A comparison of the search cost for processing an exact match query (uniform distribution)

![Figure 14](image2.png)

**Figure 14**: A comparison of the search cost for processing an exact match query (centralization distribution)

![Figure 15](image3.png)

**Figure 15**: A comparison of the search cost for processing an exact match query (diagonal distribution)

![Figure 16](image4.png)

**Figure 16**: A comparison of the search cost for processing an exact match query (X-parallel distribution)

![Figure 17](image5.png)

**Figure 17**: A comparison of the search cost for processing an exact match query (sine distribution)
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