2013 IEEE International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing

A New DHT Supporting Multi-Attribute Queries for Grid Information Services

Yawei Zhao, Fei Cai, Junjie Xie, Lailong Luo, Xiaoqiang Teng, Honghui Chen, Weijie Kong
Science and Technology on Information Systems Engineering Laboratory
National University of Defense Technology, Changsha, Hunan, P.R. China, 410073

***Abstract*—Recent structured Peer-to-Peer (P2P) systems can't be simply applied to grid information services, because grid resources need to be registered and searched using multiple attributes. This paper proposes a multi-attribute addressable DHT (MAA-DHT) for grid information services. It uses hypercube as its logical graph and provides a new resource placement mechanism based on the values of all attributes, and then supports query based on all attributes and part attributes. The join of a new object and object query operation based on all attributes can be finished within $O(\log N)$ hops for N peers. Then, this paper proposes the lookup algorithm based on global index, the corresponding query delay is still $O(\log N)$ hops. In particular, in order to maintain the topology, this paper designs a novel stabilization protocol for the MAA-DHT, which also can implement and maintain the global index within $O(2\log N)$ rounds. Furthermore, in order to save storage and query time of the global index, we propose the informed lookup mechanism based on bloom filters, the query based on part attributes still can be finished within $O(\log N)$ hops.**

I. Introduction

Grid computing is emerging as a novel approach of employing distributed computational and storage resources to solve large-scale problems in science, engineering, and commerce. Traditional centralized approaches have the inherent drawback of a single point of failure and it does not scale well to a large number of grid nodes across autonomous organizations. To overcome the above shortcomings of centralized approaches, recent structured P2P systems use message routing instead of flooding by leveraging a structured overlay network among peers. These systems typically support distributed hash table (DHT) functionality and the basic operation they offer is lookup (key), which returns the identity of the node storing the object with the key[1] and they provide very good scalability as well as failure resilience.

A. Motivation

While DHT-based structured p2p networks have some desirable properties[2][3], this kind of hash table functionality is not enough for grid information services. Grid resources typically have multiple attributes and thus need to be registered with a list of attribute-value pairs to the grid information services. Consequently, resource requesters want to be able to search resources satisfied the requirement of multiple attributes from grid information services[4][5]. Traditional DHTs don't accommodate the multi-attribute queries, thus they can't satisfy the new need of grid information service[6]. D-to-d mapping scheme such as pSearch[7] maps a d-attribute resource to a coordinate point in a d-dimensional space. Any search request is firstly routed to any point in the query region and then propagated to the remaining points in the region. Based on d-to-one mapping scheme, Midas[8] uses Hilbert space-filling curve as the d-to-one mapping function in order to index multi-attribute resources[9]. MAAN[10] addresses it by using separate instances of DHTs for each one of the attributes. However, such a solution is not efficient since it requires many replicas of same data objects in different peers, one for each of the index attributes. LORM is built on a single DHT which is called Cycloid [11], and takes advantage of its compound graph structure in the way to connect clusters by a cycle. Each cluster, consisting of several nodes, is responsible for the information of one attribute.

More recently, Literature[12] devised a static resource placement and query mechanism based on the multi-attribute value of resources for the DHT-based Systems. They introduced a tree-based order-preserved hash function, which can hash each range belonged to a static division of given attribute space to an identifier. For a multi-attribute resource, they hashed its value for each attribute to an identifier, and then obtain the final identifier via aggregating those identifiers obeyed to given mechanism. Those methods proposed in literature[12] are novel, but can't implement in fully decentralized model.

B. Contribution

This paper contains three important contributions that efficiently support multi-attribute queries in chord-like system.

First, traditional peer-to-peer system based on distributed hash function (DHT) usually places object according to its single attribute value, and just supports single-attribute based queries. In order to overcome the shortcoming, we propose a multi-attribute addressable DHT (MAA-DHT). It uses hypercube as its logical graph and provides the new object placement mechanism based on values of all attributes, and then supports query based on all attributes and part attributes. The join of new object and object query operation based on all attributes can be finished within $O(\log N)$ hops, where N denotes the number of peers in network.

Second, we propose the lookup mechanism based on global index in order to lookup object according to part attributes efficiently. In particular, we design a novel stabilization protocol for the MAA-DHT, which not only makes sure the correct topology but also implements and maintains a global index for

978-0-7695-5088-6/13 $31.00 © 2013 IEEE
DOI 10.1109/HPCC.and.EUC.2013.237

IEEE computer society

each peer within $O(2\log N)$ rounds. The query delay of our new method is still $O(\log N)$ hops.

Third, in order to save storage space and decrease query time of global index, we propose an informed lookup mechanism based on bloom filters. Each participating peer needs to maintain its finger table and another routing table whose size is just $O(\log N)$, but the query based on part attributes still can be finished within $O(\log N)$ hops.

II. Background and related work

There has been a lot of recent work on building structured overlay networks and Distributed Hash Tables (DHTs). We begin by explaining one such system, which called Chord. Then we explain a compact data structure named Bloom Filters, which will be used by our new DHT to support multi-attribute query.

Chord: Chord is a structured overlay network which allows efficient routing of a message from one node to any other node by greedy and clockwise routing. This mechanism requires only $O(\log N)$ messages for routing between any arbitrary pair of nodes. Moreover, when a node joins or leaves the system, it requires only $O(\log N)$ messages to restructure the overlay network appropriately for the new set of nodes.

Bloom Filters: A bloom filter[13] is a compact data structure for probabilistic representation of a set in order to support membership queries. A bloom filter for representing a set $S = \{s_1, s_2, \ldots, s_n\}$ of n elements is described by an vector of m bits, with all bits initially set to 0. It uses k independent hash functions $h_1, h_2, \ldots, h_k$ ranging from 1 to m. These hash functions map each item in the set to a random number uniformly over the range $1, 2, \ldots, m$. For each element $x \in S$, the bits $hi(x)$ are set to 1 for $1 \leq i \leq k$. If all $hi(x)$ are set to 1, then x is a member of S, although this would be wrong with some probability. Otherwise, we assume x is not a member of S. Hence a bloom filter may yield a false positive, for which it suggests that an element x is in S even though it is not. Each false positive is due to a bloom filter collision, in which all bits indexed previously were set to 1 by other elements[13]. The space efficiency is achieved at the cost of a small probability of false positives in membership queries. However, for many applications the space savings and short locating time consistently outweigh this drawback.

III. Multi-Attribute Addressable DHT

A. Overview

In general,three well-known families of graphs, which have been widely studied, are the hypercube, de Bruijn, and Kautz digraph[14], [15]. Although some work has been done on DHT based on de Bruijn and Kautz digraph, there is no efficient specific solution to implement it in a fully decentralized way. Chord as one of the classic DHT uses hypercube as its logical graph, but it can not support multi-attribute queries. Multi-Attribute Addressable DHT (MAA-DHT) addresses this problem by extending Chord with an object placement mechanism based on values of multi-attribute and a multi-attribute query mechanism.

The MAA-DHT also uses hypercube as the logical graph, thus each peer and object is assigned a unique ID at random from a circular m-bit identifier space $[0, 2^m)$ for an m-dimension hypercube. At its heart, MAA-DHT provides fast distributed computation of a hash function which maps objects to nodes responsible for them. It uses bloom filters and consistent hashing to map object with multi-attribute onto the identifier space. The identifier space has a distance function, and in this paper it denotes clockwise distance on the circle.

B. Mapping peers and object onto identifier space

Each participating peer receives an m-bits identifier, taken from the identifier space. To implement it, each peer p is assumed to have some unique attribute that can be used for mapping p onto identifier space. The implementation can be done in several ways. One typical approach is to select or define a globally known consistent hash function such as SHA-1, and the hashing result of p's IP address modulo 2^m is chose as the identifier of peer p. Chord and other DHTs do it according to a single attribute such as file name of object, thus they can't realize multi-attribute query on this condition. The MAA-DHT first transfer the set of each attribute value for any object as a bloom filter, and then obtain its identifier by hashing it.In general, a bloom filter can be changed into $String$, $Integer$ and other data type. Thus, the hashing of a bloom filter can be done in several ways. But different hash methods have non influence on the performance and cost of the algorithm. The identifier length m must be large enough to make the probability of two nodes or objects hashing to the same identifier negligible.

C. Interconnection between peers

Using peer identifiers and an overlay network, a directed graph is built. Nodes in this graph represent peers and outgoing arcs at a node of the graph model routing pointers that the peer should maintain. Typically, a structured p2p overlay network is built such as to guarantee logarithmic diameter while maintaining compact routing table of logarithmic or constant size.

Each participating peer p maintains a routing table with up to m entries, called the finger table. The i^{th} entry in the table at peer p contains the identifier of the first peer s, whose identifier is at least distance 2^{i-1} away from the identifier of peer p. Peer s can be called the i^{th} finger of peer p, and denoted it by $p.finger[i]$. A finger table entry includes both the identifier and access point (including the IP address and port number) of the relevant peer. The first finger of p is the immediate successor of p on the circle and is often called the successor. On the other hand, to simplify the join and leave mechanisms, each peer maintains a $predecessor$ entry which records the identifier and IP address for the immediate predecessor of that peer.

For a new peer, it needs to find the successor peer whose identifier is equal to or follows 2^{i-1} addition the identifier of peer p for each $1 \leq i \leq m$ by itself, and then initialize its $predecessor$ by other new peers. Literature[16] has proposed a

corresponding algorithm, and it is still suitable to MAA-DHT. Once all participating peers of MAA-DHT have finished their routing table, it means that the overlay network is completed.

D. Insert new object into MAA-DHT

At any point in time, each peer manages a sub-set of identifier space. Consistent hashing assigns objects to peers as follows. Each object is assigned and stored by the first peer whose identifier is equal to or follows the identifier of the object in the identifier space, and this peer is called the $successor$ peer of that object.To maintain the consistent hashing mapping when a new peer p joins the network, certain identifiers previously assigned to p's successor now are assigned to n. When n leaves the network, all of identifiers managed by it will be reassigned to p's successor.

If a participating peer p of MAA-DHT wants to add a new local object into the network, it will firstly map the new object onto the identifier space, and then transfer the object to its successor peer within $O(\log N)$ hops. Algorithm 1 give a detail description about the progress. In algorithm 1, m denotes the predefined length of identifier, and n denotes the predefined number of attributes for all participating objects.

Algorithm 1 Insert(object)

Require: object with multi-attribute is not null and Peer p inserts an object to MAA-DHT network.

```
1: Create a bloom filter(BF) of the object, and then get its id by hashing the BF
2: Placement(object.id)
3: Function: Placement(object.id)
4: if id∈ (p.identifer, p.successor.identifier) then
5:     Forwards object to the successor of current peer p, and store it locally.
6: else
7:     for each j = m downto 1 do
8:         if finger[i]∈(p.identifier,id) then
9:             Forwards request to peer p.finger[i];
10:            the target peer invokes Placement(object,id);
11:        end if
12:    end for
13: Peer p stores object locally
14: end if
```

E. Lookup based on all attributes

Any peer p may receive a lookup request which predefine the values of all attributes for target objects. If peer p can resolve this request, then it responses result and terminates this request. Otherwise, the lookup request need be forwarded to its successor peer. The mechanism of resolving lookup based on all attributes is similar to the mechanism used to insert new object into MAA-DHT. First, peer p maps the lookup request onto the identifier space, this can be done by hashing the bloom filters of the payload within this lookup quest. Furthermore, find the successor peer of the request identifier, and forward this lookup request to it within $O(\log N)$ hops. Algorithm 2 gives a detail description about the progress. In algorithm 2, m denotes the predefined length of identifier, and n denotes the predefined number of attributes for all participating objects.

Algorithm 2 Lookup(request)

Require: Request with multi-attribute is not null and Peer p receives a lookup request

```
1: if request.attributes.length =n then
2:     Create a bloom filter(BF) of the request, and then get its id by hashing the BF
3:     Forward2successor(request,id)
4: end if
5: Function: Forward2successor(request,id)
6: if id∈ (p.identifer,p.successor.identifier) then
7:     Forwards object to the successor of current peer p, and solves it locally.
8: else
9:     for each j = m downto 1 do
10:        if finger[i]∈(p. identifier,id) then
11:            Forwards request to peer p.finger[i].
12:            target peer invokes Forward2successor(request, id)
13:        end if
14:    end for
15: Peer p solves object locally
16: end if
```

IV. Lookup based on part attributes

In the previous section, we have proposed the lookup algorithm based on all attributes. In reality, upper applications also need to lookup mechanism based on part attributes not all attributes. In this section, we will propose two kinds of mechanism to resolve this kind of need.

A. Lookup mechanism based on global index

For lookup request based on part attributes, how to map request onto the identifier space becomes the critical problem. Global index can solve the problem well. Because in this mechanism,each participating peer knows information about the global distribution of all participating objects.

Global index can be implemented in several ways, such as global inverted index. In this paper, we use the identifier, IP address (including corresponding port), and the bloom filter of each participating peer as the entry of global index. For each participating peer, we use a bloom filter to present the set of each attribute value for all objects. The resulting bloom filter can be called the local index of that peer. If the local index of each participating peer can be propagated to all other peers, and then the global index will be implemented at each peer. Thus, each peer uses a copy of global index to find the successors for any lookup request based on part attributes. Furthermore, the $Forward2successor$ method mentioned in algorithm 2 can forward lookup request to each successor within $O(\log N)$ hops. Finally, each successor will resolve the lookup request locally and response query request if there is at least one satisfied object hosted locally. In order to avoid producing too many additional messages because of constructing and maintaining the global index for each peer, we will extend the stabilization protocol proposed in Chord to implement the global index at each peer.

In order to ensure that the lookups can be executed correctly as the participating peers change, any DHT must ensure that each peer's successor pointer is up to date. It does this by using a "stabilization" protocol that every peer runs periodically in the background which updates each peer's finger tables and successor pointers. The stabilization protocol includes $stabilize$ and $fix\_fingers$ major methods.The $fix\_fingers$ mentioned in Chord can't make sure its finger table entries are correct. The reason is that the $findsuccessor$ used can't find the new peer whose identifier locates between finger$[i-1]$ and finger$[i]$, thus the finger$[i]$ can't be updated correctly for $2 \leq i \leq m$. Chord periodically uses the method to identify the new joined nodes and left ones and tries to solve each query in any circumstances. Algorithm 3 gives a detail introduction about the new $Fixfingers$ method of MAA-DHT, which not only makes sure the correctness of finger table but also implements the global index. Every participating peer periodically calls $Fixfingers$ to make sure its finger table entries are correct. It is easy to know that each call only fix the next one entry according to the index parameter used by $Fixfingers$ method. Notice that $Fixfingers$ method doesn't fix the first entry, in general, which is fixed by the $stabilize$ method.

Each call of the $stabilize$ method sends the message with identifier and bloom filter of caller to its successor peer, and forward other same type messages to its successor. Each call of the Fixfingers method will propagate the local index and index of other peers received from its neighbor to all peers range over (caller.finger[index-1].identifier, caller.finger[index].identifier]. The local index of any peer will be propagated to all peers of MAA-DHT network within $O(2\log N)$ round of stabilization under the contribution of $Fixfingers$ and $stabilize$ methods. Furthermore, the changes of objects hosted by any peer also can be known by all other peers after $O(2\log N)$ round of stabilization.

Algorithm 3 Fixfingers()

Require: Peer p updates one entry of its finger table in one round of stabilization
1: **for** (**do** each $i = 1$ to m)
2: Create or obtain the index message mge with identifier and bloom filter of peer p
3: Left ← p.identifier + $2^{(index-2)}$
4: Right ← p.finger[index].identifier
5: mge.right ← right
6: $p0$ ← finger[index-1]
7: finger[index] ← p.Increment($p0, mge$)
8: Function: ***Increment(p0,mge)***
9: Peer p sends the index message mge to peer $p0$
10: Peer $p0$ propagates the local index to all peers range over (left, right]
11: middle ← $p0$.successor
12: **if** p.middle.identifier ∈ [left, right] **then**
13: return middle
14: **else**
15: return $p0$.Increment(middle)
16: **end if**
17: **end for**

In reality, controlling the stabilization rate and size of each message can decrease the overhead of stabilization. If the stabilization rate increases, the accuracy of global index and success rate of query increases at the same rate, but the resulting messages overhead also increases at the same time. There is a tradeoff between the stabilization rate and the success rate of query.

B. Informed Lookup mechanism based on Bloom Filters

The Lookup mechanism based on global index can meet the need of lookup based on part attributes, but each peer must maintain the global index with N entries and the finger table. The storage space and time used to scan the whole global index will be large. In order to overcome this problem and support the lookup based on part attributes at the same time, we propose the informed lookup mechanism based on bloom filter. For this new mechanism, each participating peer need maintain its finger table and another routing table whose size is just $O(\log N)$.

In nature, informed lookup protocol is one kind of forward-based routing protocol. It has two major components, the construction and maintenance mechanism of routing table, and a query forward mechanism using the routing table. The routing table is a set of bloom filters, and each entry corresponding to an input link. For each peer, if it receives the first index message coming from a peer, it will believe there is an input link from that peer to it and construct a routing entry for that input link. Figure 1 shows that node 0 constructs three input links with node 1, node 2 and node 5. The initialized bloom filter for that link is assigned using the bloom filter enveloped in the received index message. The ongoing index message received from the same input link will update the bloom filter, this can be done by a logical or operation between the bloom filter of that input link and the bloom filter enveloped in the ongoing index message.

The lookup mechanism based on part attributes is tightly associated with the bloom filter corresponding to each input link. When a peer issue a lookup based on part attributes, that peer will try to resolve it itself. Furthermore, the bloom filters corresponding to each input link of that peer will be scanned and desired links will be filtered out as the candidate input links, finally the lookup request will be forward to the inverted direction of candidate input links. Figure 4 shows that if there exists a part attribute query in the node 0. It solves it itself and then checks which input links can result in more results. The peers received those lookup request will try to resolve it by those its local objects, and also forward that lookup request to all candidate directions if there is at least one desired input link. Algorithm 4 gives a detail description about the informed lookup progress.

For any pair of source and sink peer, there exists only one path by which the index message of source peer can be forwarded to sink peer. This fact can be guaranteed by algorithm 3. In other word, for lookup aimed at the objects stored by the source peer, there is only one forward path from the sink peer to the source peer, and the length upper bound

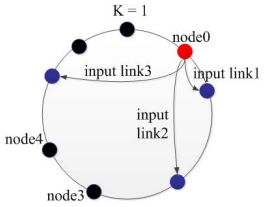

Fig. 1. Three input links are constructed by informed lookup protocol

is $2 \log N$. That means that any lookup based on part attribute issued from any peer will be resolved by all candidate peers within $O(\log N)$ hops.

There is false negative (The lookup mechanism believes that none peer can meet the constraint of lookup, even there exists at least one peer which can) if a node issues a lookup based on part attributes before the index message of desired peers has propagated over the whole p2p network. This problem is also faced by the lookup mechanism based on global index. The false negative can decrease if the stabilization rate improves, because more frequent stabilization operation can make sure all peers obtain the out-of-data index message about other peers. It is also need to make decision about the tradeoff between overhead and accuracy.

Algorithm 4 InformedLookup(query)

Require: peer p issues a lookup query based on part attributes
1: **for** each $i = 1$ to inputlinks.length **do**
2: **for** each $j = 1$ to query.attributes.length **do**
3: BF $\leftarrow$ inputlinks.[i].bloomfilter
4: **if** BF.Query(query.attributes[j]) = false **then**
5: break;
6: **end if**
7: Peer p forward query to peer whose identifier is inputlinks.[i].identifier
8: **end for**
9: **end for**

V. Simulation

MAA-DHT is verified by finishing lookup request successfully and measuring its performance. The implementation uses sockets to communicate between the peers, and new object can be added by contacting any existing peer at its IP address and port number.

It can easily be configured to support different attribute schemas. We choose the book information as an example. Book information contains name, author and publishing house, which can be viewed as the attributes of a book. The entire experiment includes checking the correctness of lookup mechanism based on the whole attributes and part attributes, and analysing the efficiency difference of multi-attribute queries between MAA-DHT and MAAN as well as the false positive of bloom filter.

We establish a Chord model with 100 nodes, which stores 10,000 objects and each object uses three strings to represent its properties. The object ("Brief History of Time", "Hawking", "Bantam Dell Publishing Group") can be converted to a bloom filter. Then we establish a global index list to store all local indexes and then broadcast the copy of global index to others. After the whole process, a query request which contains part attributes "Brief History of Time" and "Hawking" is generated. The lookup mechanism is proved to be correct because we can find the object successfully

In fact, the object ("Brief History of Time", "Hawking", "Bantam Dell Publishing Group") is only stored in one node. But two nodes are chosen as the responded node. The reason is that the bloom filter has a false positive. In this experiment, the number of the hash function is 5, and the number of bits for the bloom filter is 3000. The unwanted node has stored 429 objects and the number of element, which denotes its local index, reaches 1287. Its false positive reaches 0.536, and it is too large for use. One reason is that node identifiers do not uniformly cover the entire identifier space[16]. The consistent hash paper solves this problem by associating keys with virtual nodes, and mapping multiple virtual nodes (with unrelated identifiers) to each real node. Ideally, each node stores 100 objects and false positive is 9.43×10^{-3}.

MAAN proposes two mechanisms to solve the multi-attribute query. $IterativeQueryResolution$ addresses it by using separate instances of DHTs for each one of the attributes. This solution is easy to implement, but its performance is bad, and it costs too much storage place. In fact, it takes $O(M \log N)$ routing hops to solve the query. M denotes the number of attributes. $SingleAttributeDominatedQueryResolution$ improves the performance of $IterativeQueryResolution$, which can finish the process in $O(\log N)$ routing hops, and saves part of storage place, but it also needs to replicate the object for each one of the attributes. However, lookup mechanism based on all attributes takes $O(M \log N)$ routing hops to solve the query, and does not cost much additional storage place.

We consider a network with 10000 multi-attribute objects, and vary the total number of nodes from 2 to 1024. For each value, we repeat the experiment 100 times. Figure 2 plots the mean and the minimum and maximum of the routing hops for queries. As expected, the measured average routing hops only increase logarithmically with the number of nodes, and it roughly match with our theoretical analysis. Bloom filter has false positive and it may cause some wrong results. When a node issues a part attribute query, it transforms the query request to a bloom filter and then checks the global index to find the whole nodes which may contains the objects we need. However, some nodes are called the bad routings which are meaningless and cost extra time because of the false positive of bloom filter. The percentage of bad routing has a significant impact on the performance of query because of the false positive of bloom filter.

Figure 4 shows the percentage of bad routing over a range of the numbers of bits for the bloom filter. Experiments with larger than one thousand bits for the bloom filter are conducted by running the model 10 times independently. For each number of bits shown in Figure 4, there are 100 part-attribute queries and the graph plots the average of percentage of bad routing in the whole lookups. It can be seen from Figure 4 that the

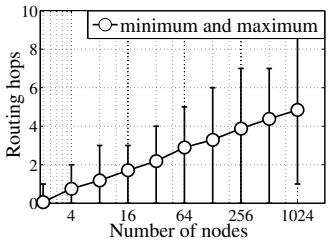

Fig. 2. The number of routing hops is a logarithmic function of network size

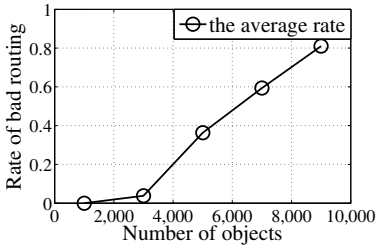

Fig. 3. The percentage of bad routing increases with the number of objects

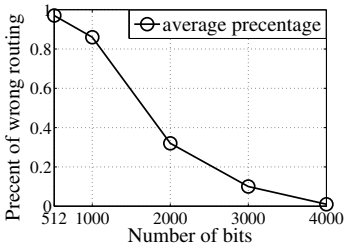

Fig. 4. The percentage of bad routing decreases with the number of bits

percentage of bad routing decreases with number of bits for the bloom filter obviously. It is an effective way to decrease the bad routing by increasing the number of bits appropriately.

MAA-DHT uses a bloom filter to identify one specular object. It hashes all the attributes of an object to a bloom filter and then gets the identifier by hashing the bloom filter. Figure 3 shows that the rate of bad routing increases as the number of objects stored in the node. It is easy to understand because if one node stores more objects, the more hash functions is used to get the bloom filter, and the more faults will be made. MAA-DHT uses the Chord-like model and avoids the problem by "virtual node"[16].

We has known the MAA-DHT can implement multi-attribute queries by bloom filters. It creates one bloom filter for a multi-attribute object by hashing each of the object's attributes. No matter how many attributes the object has, MAA-DHT can finish the query by costing a certain amount of storage space.

VI. Conclusion

This paper proposes a new distributed hash table named multi-attribute addressable DHT supporting multi-attribute queries. It provides a new object placement mechanism based on values of all attributes, and then support query based on all attributes and part attributes. The join of new object and object query operation based on all attributes can be finished within $O(\log N)$ hops for N peers. Then, this paper proposes the lookup algorithm based on global index for those query based on only part attributes, the corresponding query delay is still $O(\log N)$ hops. In particular, we design a novel stabilization protocol for the MAA-DHT to implement and maintain the global index within $O(2\log N)$ rounds. Furthermore, in order to save storage and query time of the global index, we proposes an informed lookup based on bloom filters, the query based on part attributes still can be finished within $O(\log N)$ hops.

In the future, we will design and implement new distribute hash table supporting range and multi-attribute queries together. Furthermore, we will study other scalable data structures and distributed spatial index, and design new structured peer-to-peer network.

Acknowledgment

This work is funded by the NSFC under Grant No. 61070216 and the Hunan Provincial Innovation Foundation for Postgraduate under Grant No.B130503.

References

[1] S. Ratnasamy, I. Stoica, and S. Shenker, "Routing algorithms for dhts: Some open questions," in *Peer-to-Peer Systems*, ser. Lecture Notes in Computer Science, P. Druschel, F. Kaashoek, and A. Rowstron, Eds. Springer Berlin Heidelberg, 2002, vol. 2429, pp. 45–52.

[2] D. Guo, H. Chen, and X. Luo, "Resource information management of spatial information grid," in *In: LNCS3032*. Springer, 2003, pp. 240–243.

[3] C. Yang, D. Guo, Y. Ren, X. Luo, and J. Men, "The architecture of sig computing environment and its application to image processing."

[4] D. Guo, J. Wu, H. Chen, and X. Luo, "Moore: An extendable peer-to-peer network based on incomplete kautz digraph with constant degree," in *IN: PROC. 26TH IEEE INFOCOM, IEEE COMPUTER SOCIETY PRESS, LOS ALAMITOS*, 2007.

[5] D. Guo, H. Chen, Y. Liu, and X. Li, "Bake: A balanced kautz tree structure for peer-to-peer networks," in *In Proc. 27th IEEE INFOCOM*, 2008.

[6] Y. Yuan, D. Guo, G. Wang, and L. Chen, "Removing uncertainties from overlay network," in *Database Systems for Advanced Applications*, ser. Lecture Notes in Computer Science, J. Yu, M. Kim, and R. Unland, Eds. Springer Berlin Heidelberg, 2011, vol. 6587, pp. 284–299.

[7] C. Tang, Z. Xu, and S. Dwarkadas, "Peer-to-peer information retrieval using self-organizing semantic overlay networks," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM '03. New York, NY, USA: ACM, 2003, pp. 175–186.

[8] V. March and Y.-M. Teo, "Multi-attribute range queries on read-only dht," in *Computer Communications and Networks, 2006. ICCCN 2006. Proceedings.15th International Conference on*, 2006, pp. 419–424.

[9] Y. LAN and H. DENG, "A comparative study of three kind of multi-attribute range query based on dht," *Journal of Computational Information Systems*, vol. 8, no. 6, pp. 2657–2669, 2012.

[10] M. Cai, M. Frank, J. Chen, and P. Szekely, "Maan: A multi-attribute addressable network for grid information services," *Journal of Grid Computing*, vol. 2, no. 1, pp. 3–14, 2004.

[11] H. Shen, C. Xu, and G. Chen, "Cycloid: A scalable constant-degree p2p overlay network," *Performance Evaluation*, vol. 63, no. 3, pp. 195–216, 2012.

[12] D. Li, X. Lu, B. Wang, J. Su, J. Cao, K. Chan, and H. va Leong, "Delay-bounded range queries in dht-based peer-to-peer systems," in *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, Lisboa, Portugal, 2006, pp. 64–64.

[13] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, pp. 422–426, 1970.

[14] D. Guo, Y. Liu, and X. Li, "Bake: A balanced kautz tree structure for peer-to-peer networks," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, Phoenix, AZ, USA, 2008, pp. 2450–2457.

[15] D. Guo, J. Wu, Y. Liu, H. Jin, H. Chen, and T. Chen, "Quasi-kautz digraphs for peer-to-peer networks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 6, pp. 1042–1055, 2011.

[16] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *Networking, IEEE/ACM Transactions on*, vol. 11, no. 1, pp. 17–32, 2003.