Balanced Kautz Tree for Delay Bounded Range Query Processing in Peer-to-Peer Network

Prof. Chandrakant Umarani, Prof. Sanjay Ankali
Asst. Professor, KLECET, Chikodi

Abstract - With the increasing popularity of the peer-to-peer (P2P) computing paradigm, many efficient range query schemes for distributed hash table (DHT)-based P2P systems have been proposed in recent years. Although those schemes can support range query without modifying the underlying DHTs, they cannot guarantee to return the query results with bounded delay. The query delay in these schemes depends on both the scale of the system and the size of the query space or the specific query. In this paper, we propose an efficient range query processing scheme to support delay-bounded single-attribute and multiple-attribute range queries. We first describe the order-preserving naming algorithms for assigning adjoining ObjectIDs to objects with close attribute values. Then, we present the design of the forwarding tree to efficiently match the search paths of range queries to the underlying DHT topology. Based on the tree, two query processing algorithms are proposed to, respectively, process single-attribute and multiple attribute range queries.

Index Terms—Peer-to-peer computing, distributed hash table (DHT), delay bounded, range query.

1 INTRODUCTION

In this paper, we describe an efficient delay bounded efficient range query scheme. Efficient range query scheme operates on top of a high-performance constant-degree DHT scheme, and does not need to modify the underlying DHT infrastructure. DHT provides support for scalable and efficient exact-match query of distributed objects on peers. However, it cannot support range queries for attribute values. The basic components of efficient range query scheme include two parts: order-preserving naming and range query processing. Efficient range query scheme first uses order-preserving naming algorithms to assign to objects with close attribute values the Object IDs adjoining in the Kautz namespace so as to publish them on related peers. Then Efficient range query scheme provides efficient query processing algorithms to forward range queries to appropriate peers and return query results within a bounded delay. Based on the number of attributes that queried, range queries can be classified as the single-attribute range query and the multiple-attribute range query. Efficient range query scheme adopts order-preserving naming algorithms Single hash and Multiple hash, and query processing algorithms PIRA and MIRA, to perform single-attribute and multiple-attribute range queries, respectively.

Efficient range query scheme is built on top of the underlying DHT. It relies on the underlying DHT to organize its P2P overlay and provide much of the robustness, availability, and load balancing. Efficient range query scheme uses the naming algorithms to assign to objects order-preserving Object IDs and efficiently propagates the range queries in the overlay, while the underlying DHT organizes the peers in an overlay and handles the dynamic joining or leaving of peers. If a peer fails, the underlying DHT automatically ensures that other peers in the overlay takes over the responsibility for the failed peer and provides graceful fail-over by using replication or other mechanisms. And the underlying DHT also deals with the routing and publishing of objects according to the Object IDs. In some sense, the underlying DHT shields Efficient range query scheme from the dynamics of peers and the complexity of the P2P overlay, so the design of Efficient range query scheme can be focused on the naming and range query processing algorithms.

2 RELATED WORK

The main contributions of this paper include the following three parts:

1. We propose the partition tree model to provide order preserving mappings from the query space to the namespace of DHT. The single-attribute naming algorithm Single hash and the multiple-attribute naming algorithm Multiple hash are designed to assign adjoining ObjectIDs to objects with close attribute values, so that they can be published to the same or related peers in the system to support efficient range queries.

2. We design the forward routing tree (FRT), which matches the search paths of range queries to the underlying DHT topology efficiently. Based on the
we propose the range query processing algorithms Pruned Routing Algorithm (PIRA) and Multiple-attribute pruning Routing Algorithm (MIRA) to, respectively, perform single-attribute and multiple-attribute range queries within a bounded delay.

3. We analyze the lower bounds on the delay and message cost for range queries, and evaluate the query delay and message cost of Armada by both theoretical analysis and simulations. For general range query, the scheme first routes the query to the peer in charge of the PeerIDs, and first it checks whether PeerID is prefix of ObjectID, if so then forwards the query to its related peer. An efficient indexing structure called BK (balanced Kautz tree) tree that uniformly maps the m-dimensional data space onto DHT nodes, and then proposes a BK tree-based range query scheme called ERQ that processes range queries in a parallel fashion and guarantees to return the results in a bounded delay. ERQ shows that the BK tree is an efficient indexing structure for distributed complex queries.

In DHT, the identifiers of peers (i.e., PeerIDs) are Kautz strings of base 2 and their lengths may be different. The maximum length of PeerIDs is less than \(2 \log N\) and the average length is less than \(\log N\). Peers are organized into an approximate Kautz graph according to their PeerIDs. Each object in DHT is assigned an ObjectID by the naming algorithm Kautz_hash. ObjectIDs are Kautz strings distributed in the Kautz namespace KautzSpace(2,100) and are of fixed length 100, which is long enough for a P2P system with \(2^{50}\) peers. Each object is published onto a Unique peer whose PeerID is a prefix of its ObjectID.

The scheme compares the directed controlled flooding (DCF) mechanism (hereafter called DCF-CAN) which can achieve a good overall performance with Efficient range query scheme but it has a query delay of more than \(O(N^{\Delta/d})\), with an increasing rate almost proportional to the increase in the size of range queries. DCF-CAN can support only single-attribute range query. Wherein efficient range query scheme has average query delay less than \(\log N\).

3 SYSTEM ARCHITECTURE

It first uses order-preserving naming algorithms to assign to objects with close attribute values the objectIDs adjoining in the Kautz namespace, so as to publish them on related peers.

<table>
<thead>
<tr>
<th>Applications</th>
<th>Efficient Range Query Scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>App 1</td>
<td>Single Attribute query processing</td>
</tr>
<tr>
<td>App 2</td>
<td>Multi Attribute query processing</td>
</tr>
<tr>
<td>Appn</td>
<td>Single attribute naming (Single Hash)</td>
</tr>
<tr>
<td></td>
<td>Multi attribute naming (Multi Hash)</td>
</tr>
</tbody>
</table>

Fig.1 System Architecture

Then, provides efficient query processing algorithms to forward range queries to appropriate peers and return query results within a bounded delay. Based on the number of attributes that queried, range queries can be classified as the single-attribute range query and the multiple-attribute range query. Range query scheme adopts order-preserving naming algorithms Single hash and Multiple hash, and query processing algorithms PIRA and MIRA, to perform single-attribute and multiple-attribute range queries, respectively. DHT organizes the peers in an overlay and handles the dynamic joining or leaving of peers. If a peer fails, the underlying DHT automatically ensures that other peers in the overlay takes over the responsibility for the failed peer and provides graceful fail-over by using replication or other mechanisms. And the underlying DHT also deals with the routing and publishing of objects according to the ObjectIDs.

4 MODULES

Forward tree construction (FRT)
Partition tree construction.
Identifying peer region
Forwarding range query

4.1 FORWARD TREE CONSTRUCTION
In forward tree construction, peerID should be assigned for each peer and their length of each peerID may be different. Neighboring symbols in each peer should be different.

FRT is formed by using following rules:
1. The root is peer P.
2. Each node in the FRT is a peer in DHT.
3. For each node in FRT, its child nodes at next level is its neighbours.
4. The FRT has (n+1) levels with root node at 0th level.

4.2 PARTITION TREE CONSTRUCTION

Partition Tree
Assign edge values (0,1,2)
Assign interval [L,H] into root node
Split into interval and sub interval
Identify peer region

Fig. 3 Partition Tree Construction
Partition tree construction is similar to binary tree. Partition tree has n+1 level with root node at the 0th level. The root node has three child nodes, while other intermediate nodes have only two children. Labels of edges from a father node to its children can be 0 or 1 or 2, increasing from left to right, but they should be different from in-edge’s label of the father node.

We partition the entire interval of attribute values [L, H] onto the partition tree. The root node represents the entire interval [L, H] and other nodes represent subintervals of [L, H]. Each child node evenly partitions the subinterval represented by its father node.

4.3 IDENTIFYING PEER REGION

In this module, single_hash naming algorithm is designed to be an interval-preserving function from attribute-value interval [L, H]. Any attribute-value subinterval can be mapped to a Kautz region in the charge of some related peers and identify the peer which handles range query. Then, range queries can be performed by forwarding queries to the appropriate peers.

4.4 FORWARDING RANGE

Fig. 5 Forwarding range query
In this module, Based on the FRT, Armada perform a search in the FRT for all the destination peers that are in charge of the Kautz region [LowT, HighT]. Suppose the Kautz strings LowT and HighT have a common prefix, then forward query request to all the destination peers are at the same level of the FRT. Query result is returned within bounded delay.

5 DATABASE DESIGN

Database-backend applications use databases in very specific ways. They do all the input, processing, and display in the application. They use the database to store information that must be kept after the application exits and information that must be shared with other applications.

Fig.6 Database Design
In summary, the application does its own
- Input
- Processing
6 METHODOLOGIES

The design of efficient range query scheme can be focused on the naming and range query processing algorithms respectively, for single-attribute and multiple-attribute range queries.

6.1 SINGLE-ATTRIBUTE RANGE QUERY

In this section, we present the design of the single-attribute range query scheme in efficient range query scheme.

6.1.1 Single-Attribute Naming

We propose an order-preserving naming algorithm Single hash to assign to objects with close single attribute values the ObjectIDs adjoining in the Kautz namespace. According to the properties of DHT, objects with adjoining ObjectIDs are published on the same or related peers.

Algorithm on single_hash

{  
Single_hash (attributevalue c, value L, value H, length k)  
Step 1 : Initialize left to 0 and right to 1.  
Step 2 : Initialize nextid[0][left] to 1, nextid[0][right] to 2.  
Step 3 : Initialize nextid[1][left] to 0, nextid[1][right] to 2.  
Step 4 : Initialize nextid[2][left] to 0, nextid[2][right] to 1  
Step 5 : if c>H or c<L then return error.  
Step 6 : initialize S to null.  
Step 7 : if c ∈ [L,L+1/3*(H-L)] then goto Step 8.  
Step 8 : Assign 0 to S[0],a<L,b<L+1/3*(H-L).  
Step 9 : if c ∈ [L+1/3*(H-L),L+2/3*(H-L)] then  
goto Step 10.  
Step 10: Assign 1 to S[0],a<L+1/3*(H-L), b-<L+2/3*(H-L).  
Step 11 : if c ∈ [L+2/3*(H-L),H] then goto Step 12.  
Step 12: Assign 2 to S[0],a<L+2/3*(H-L),b-<H  
Step 13 : for i<-1 to k-1 do  
Step 14 : if c>(a+b)/2 then do  
Step 15 : assign direction<-right,a-<(a+b)/2 else goto step 16  
Step 16: assign direction<-left,b-<(a+b)/2.  
Step 17 : assign S[i]<-nextid[S[i-1]][direction].  
Step 18 : return S and end.  
}

We propose a partition tree P (2, k) model to help design the Single hash algorithm. We partition the entire interval of attribute values (L, H) onto the partition tree P (2, k). The root node represents the entire interval (L, H) and other nodes represent subintervals of (L, H). Each child node evenly partitions the subinterval represented by its father node. In the example shown in Fig. 7, the attribute value 0.1 is in the subinterval represented by the leaf node P with label 0120; thus, the Kautz string 0120 is assigned as the ObjectID of the object with attribute value 0.1 by the Single hash algorithm.

![Partition tree P (2, 4) for attribute-value interval [0, 1].](image)

Algorithm on PIRA:

PIRA (value low V, value high V);  
Step 1: if low V<high V then return error  
Step 2: assign lowT<single_hash (lowV , L, H, k) and  
Step 3: assign highT<single_hash (highV , L, H, k).Based on the  
Step 4: assign comT<commonprefix (lowT, highT).  
Step 5: if comT=null then goto step 6  
Step 6: assign rangeset<dividerange (lowT, highT);  
Step 7: for each rangei € rangeset goto step 9  
Step 8: do Punningsearch (rangei, lowT, rangei, highT) else goto step 9  
Step 9: Punningsearch (lowT, highT); end;  
Algorithm on Punningsearch
{\text{Prunningsearch (string lowT, string highT)}}
Step 1: assign comT<commonprefix (lowT, highT);
Step 2: if is prefix (p, comT) then query (p)
Step 3: comS<suffixprefix (peerid (p), comT);
Step 4: maxlevel<lengthof (peerid (p)-lengthof comS);
Step 5: Idown (maxlevel, lowT, highT); end;
Algorithm on Idown

{Idown (depth h, string lowT, string highT)}
Step 1: if h=0 then goto step 2
Step 2: then query (u), return;
Step 3: else for each R=a2……ahXY € outneighbors (u)
Step 4: do W<X-Y;
Step 5: len=lengthofstring (W);
Step 6: if (prefix (lowT, len) <W) and (W<prefix (highT, len))
Step 7: R.Idown (h-1, lowT, highT);

Fig.8 An example of FRT.

Fig. 9 shows an example of using PIRA for search in the FRT shown in Fig. 8. In the example, peer 212 issues a range query [0.1, 0.24], the entire attribute-value interval is [0, 1] and k =4. By the Single hash algorithm, we can get LowT = 0120 and HighT = 0202. Thus, the destination peers are all at the third level of the FRT. The dashed lines with arrows in Fig. 8 show search paths of PIRA.

6.2 MULTIPLE-ATTRIBUTE RANGE QUERY
Many applications require the support for multiple-attribute range query on DHTs, e.g., the query “15 _ age _ 18 and 80.5 _ score _ 95”.

6.2.1 MULTIPLE-ATTRIBUTE NAMING
We use the partition tree to help design the multiple-attribute naming algorithm, Multiple hash, to assign to objects partial-order preserving ObjectIDs. We partition the entire multiple-attribute space < [L0, H0], . . ., [Li, Hi], . . ., [Lm-1,Hm-1]> onto the partition tree along attributes A0,A1, . . ., and Am-1 in a round-robin style. Each node in the partition tree represents a multiple-attribute subspace and the root node represents the entire space < [L0, H0], . . ., [Li, Hi], . . ., [Lm-1,Hm-1]> . For any node P at the jth level with f child nodes, let i denote the value of j modm. Then, the subspace w represented by node P is evenly divided into f pieces along the ith attribute (i.e., attribute Ai), and each of the f child nodes represents one piece Thus, each node at the same level of the tree represents a multiple-attribute subspace of the same size and the union of all such subspaces is the entire multiple-attribute space. Fig. 10 shows an example of the partition tree P(2,4) that represents the 2D (m = 2) multiple-attribute space <[0, 6], [0, 8]>.

It works as follows: For any object O with the multiple attribute value V =< v0, v1, . . ., vm-1 >, V is surely in a subspace represented by a leaf node in the partition tree. Then, the label of the leaf node is assigned as O’s ObjectID.

6.2.2 Multiple-Attribute Range Query Processing
We propose a new algorithm, called MIRA, to process multiple-attribute range queries. MIRA...
follows the basic idea of PIRA to perform pruning search on the FRT of peer \( P = u_1 u_2 \ldots u_b \) that issues the range query \( w \). In the example in Fig. 10, we set \( m = 2 \), \( k = 4 \) and the multiple-attribute space is \( [0,6], [0,8] \) peer 212 issues a multiple-attribute range query \( < [1.2,1.8],[1.5]> \). By the Multiple hash algorithm, we can get LowT =0120 and HighT =0210. Therefore, the destination peers are all at the third level of the FRT.

The dashed lines with arrows in Fig. 11 show the search paths of MIRA. The search message is not forwarded to peer 2020 because there is no intersection between its descendants and the destination peers.

![Fig.11 An example of MIRA.](image)

7 RESULTS
Among the well-known efficient range query schemes, only PHT, DCF-CAN, and Efficient range query scheme can support single-attribute range queries and multi-attribute range queries on constant-degree DHTs. Since the delay and message cost of PHT is much larger than that of Efficient range query scheme, we only compared the single-attribute range query scheme and multi-attribute range queries of Efficient range query scheme with DCF-CAN when the degree of the underlying DHT is equal (i.e., the parameter \( d \) is set to be 2 in DCF-CAN).

The DCF-CAN scheme uses CAN as the underlying DHT. When a peer \( P \) invokes a range query \([l, u]\) in DCF-CAN, it first routes the query to the peer in charge of the median value and then starts two “waves” of propagation. In the first wave, the current peer propagates the query only to the neighbors that intersect the query and have a “higher” interval than the current peer. Then, the current peer propagates the query to the neighbors with a “lower” interval.

![Fig.12 The impact of range size on query delay](image)

![Fig.13 The impact of network size on query delay](image)

![Fig.14 The impact of network size on message cost](image)
CONCLUSION

In this paper, we will have two independent modules as shown below. We proposed a delay-bounded range query scheme, called efficient range query scheme. Built on top of a high-performance constant-degree DHT, efficient range query scheme supports single-attribute and multiple-attribute range queries. Both analytical and simulation results demonstrated that efficient range query scheme is delay-bounded and highly efficient. The average query delay is less than \( \log N \) and the maximum delay is less than \( 2 \log N \), independent of the size of query space and specific queries. The average message cost of single-attribute queries is about \( \log N + 2n - 2 \) (\( n \) is the number of peers that intersect with the query), which is very close to the lower bound on message cost of range queries on constant-degree DHTs. Furthermore, we are extending efficient range query scheme to support attribute values in various forms and provide other complex query capabilities such as the top-k query and fuzzy query.

REFERENCES

[1] Yiming Zhang, Ling Liu, Xicheng Lu and Dongsheng Li, —Efficient range query processing over DHTs based on the balanced Kautz tree 2010.