Efficient Indexing Scheme over DHT

Yi Yi Mar
University of Computer Studies, Yangon
yiyimar.yym@googlemail.com

Khine Moe Nwe
University of Computer Studies, Yangon.

Aung Htein Maw
University of Computer Studies, Yangon.

Abstract

Range query processing is an essential role in many P2P applications including database indexing, distributed computing and location aware computing and commercial services based on current user's location. Most of P2P applications are running over DHT overlay network. But most of DHT can only provide for exact match queries. If there is no an efficient indexing scheme over DHTs, complex queries such as range queries may be major challenge. So in recent years, there are many indexing schemes over DHTs are proposed for range query processing. In this paper, an efficient indexing scheme is proposed for multidimensional range query processing. For building the proposed scheme, data are firstly distributed over peer nodes. For fairly distributing of data, k-dimensional tree (kd-tree) is used. While partitioning data on kd-tree, splitting points with each dimension are generated. Leave nodes of kd-tree store data records with tree information including splitting points. The proposed indexing scheme is built using these stored tree information. By using the proposed system, bandwidth and time consuming in range query processing over DHTs can reduce.

Key words: Indexing over DHT, range query processing, range query in P2P systems.

1. Introduction

Nowadays peer-to-peer systems are more popular in many distributed applications such as file and resource sharing. DHT is designed for structured P2P overlay network. DHT has no centralized control to reconfigure due to the churn rate of nodes. More popular DHTs based peer-to-peer systems are Chord [4], Pastry [5], Tapestry [6] and CAN [7]. There are many researches proposed over-DHT indexing approaches and DHT-dependent indexing approach. The latter can provide better query performance but it needs to be changed according to the overlay DHT infrastructure. Over-DHT indexing scheme can run over any DHTs overlay structure with no needs to modify.

DHT is robust against node failures. It is also scalable due to the properties of easily accommodate new nodes. DHT is high efficient in exact match queries in most applications but can be inefficient in manner of keyword queries and range queries. This is because it can't rely on the structural properties of key space and of the use of uniform hashing. DHT uses cryptographic hash functions like SHA1.Due to this uniform hashing method; range query processing highly desired in many p2p applications becomes an issue for look up over DHT.

This paper proposed the efficient indexing scheme built over DHT to handle the issue of lookups in DHT for range queries.

2. Related Work

There are many proposed approaches to construct indexing scheme for range query processing. Prefix Hash Tree [PHT] [1] is a distributed data structures that enables more sophisticated query over DHT. For efficiently processing 1-dimensional query over DHT, it implemented trie-based distributed data structure called Prefix Hash Tree (PHT).In range query processing, PHT proposed two algorithms. The first algorithm resulted in high latency as all leaves are sequentially traversed until the query is completely resolved. In second algorithm, it is parallelized and recursively forward the query until the leaf nodes overlapping the query. It also used binary search. But it may lead over loading the root when range is small.

Filling internal node with data can violate traversing down to leaf node. So Distributed Segment Tree [DST] [2] is designed to allow internodes to store keys as well as leaf nodes. To process a range query, at first it is decomposed into a union of minimum node intervals of segment tree. Finally the query is resolved by the union of keys returned from the corresponding DST nodes. Here, it may leads maintenance overhead as key are replicated over internal nodes and leave nodes.

Distributed Hilbert R-trees (DHR-trees) [8] provides range query processing structure for P2P systems. It can make fault tolerance and scalable to dynamic network. It can achieve efficient range query processing. But in its range query processing, first it reduce the m-dimensions to one-dimensions.

In [3], m-LIGHT can achieve efficient range query result. It used three mechanisms to construct the indexing structure over DHTs. m-LIGHT is high efficient in range query processing but it still have drawback of bandwidth and latency. This drawback is based on the lookup operation. If we make more lookups on DHT, the more the bandwidth and latency is consuming.

In this paper, the main goal is to reduce the number of DHT lookups. By reducing the lookups, the bandwidth and latency can be reduced.

3. Range Query Processing

Range query processing is finding all the data in a region in a given key range. In ran query processing applications, data sets are mostly in the form of points. For efficient range query processing, the collection of points in space and locating the data points to peer nodes with optimal load balance are major challenges. Without efficient indexing scheme, query processing is time consuming. In recent year, many researchers have proposed various indexing schemes over existing DHTs using data structures such as trees, grids and graphs to solve the above indexing problem.

In this paper, the proposed scheme also uses a data structure called k-dimensional tree (kd-tree) for constructing indexing scheme over underlying DHTs. Kd-tree is a space partitioning data structure for organizing points in a k-dimensional space. They are useful in several applications, such as searches involving a multidimensional search key including range query [11] [9].

4. Optimal Load Balance

Load balancing is an essential functionality to provide fair load distribution between peer nodes. In this proposed system, a threshold value represented as T_{pt} is used for load distribution among nodes. T_{pt} is the maximum number of loads or records that a peer node can store. Use

(1), value of T_{pt} can get, and where T_r is the total number of records in the system and N is the total number of nodes in overlay network.

$$T_{pt} = T_r / N \quad (1)$$

5. System Overview

Consider a set of data records. Each data record has a key. Each key has multidimensional data which are represented by points. Each point is in interval [0, 1]. Before constructing multidimensional data indexing scheme, data keys are firstly partitioned on kd-tree. After building kd-tree, each leaf node of kd-tree is needed to map to each peer.

Next section described how to build kd-tree for partitioning data points.

5.1 Partitioning Data on Peers

For indexing multidimensional data, the data are recursively partitioned into cell along with different dimensions in an alternative fashion. As shown in figure 2, the 2D space is recursively halved along x and y axes, alternatively until each cell contain no more than T_p in (1) data records.

In this proposed system, the splitting points are stored on internal nodes. The proposed indexing scheme is based on these splitting points. Here one important point is that the data keys are only stored in leaves. The leaves also store the tree information.

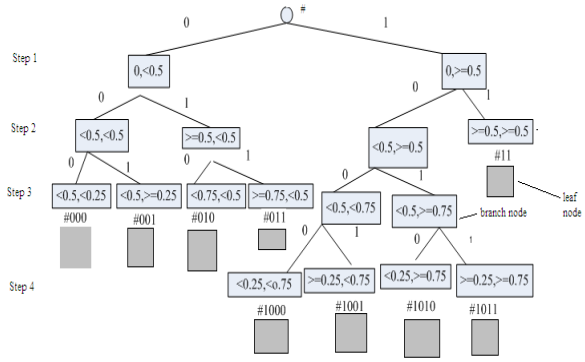
Figure 1 shows that an algorithm how to build kd-tree. In this algorithm, the parameter T_{pt} in (1) $_{\text{indicates}}$ the maximum number of data points that each cell holds. Let v be the root node and v_{left} be the left sub tree of v and v_{right} be the right sub tree of v. In algorithm, BuildKdTree (P, flag), where P is the set of data points and number of data points in P must be >=1.

Algorithm: BuildKdTree (P, flag)

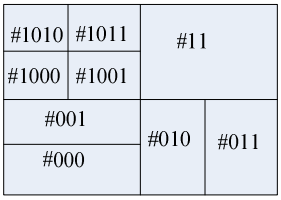
1. if P contain only one point
   then return a leaf storing this point
2. if (flag= 'y' and number of points in P $<T_{pt}$) then Split P into two subsets P1 and P2 . Let P1 be the set of points <= median point of y coordinate. Let P2 be the set of points > median point of y coordinate. Each internal node stores the median point (splitting point). Then assign flag= 'x'.
3. else if(number of points$<T_{pt}$) Split P into two subsets P1 and P2 . Let P1 be the set of points <= median point of x coordinate. Let P2 be the set of points > median point of x coordinate. Each internal node stores the median point (splitting point). Then assign flag= 'y'.
4. v_{left}=BuildKdTree(P1,flag)
5. v_{right}=BuildKdTree(P2, flag)
6. return the address of the root v with v_{left} and v_{right}

Figure 1. Kd-tree algorithm

Parameter "flag" is used to indicate the algorithm needs to partition the data points on kd-tree according to which dimension. The data points are stored only in leave node and the internal nodes store the pair of splitting points of the two dimensions.



(a) kd-tree



(b) Space partitioning

Figure 2. Space partition with kd-tree

As shown in Figure 2 (b), the set of data points are recursively partitioned into each cell along different dimensions of x axis and y axis alternatively. This partitioning process is terminated when a cell contains no more than T_{pt} data records. This space partition approach renders the local space indexed by each node to be known globally.

This space partition tree is shown in (a) as a kd-tree with labels. The root node is labeled with "#". The nodes in kd-tree are labeled with sequence of binary digits "0" and "1". As the above figure shows that the left nodes of the root are labeled with labels of its ancestors plus "0", and the right edges with plus "1".Each of the internal nodes stores splitting points established when partitioning the data on peers for achieve efficient indexing scheme. The data points are only stored in leaves.

On each leaf, leaf bucket, a distributed data structure is used for storing label of leaf node, kd-tree summarization, and data records. In each peer node, kd-tree summarization contains all the internal nodes with splitting point's information and leaf node of its own. Here the indexing scheme needs to map label of leaf node with data records to underlying DHT.

Next section discussed how to map the leave nodes of kd-tree to peers of DHT overlay network.

5.2 Mapping to DHT

To get the required data in a given query range, the result of the proposed indexing scheme is applied over underlying Distributed Hash Table (DHT) which holds (key, value) pairs. DHTs are distributed over peer nodes. So each peer needs to check if it has the requested (key, value) pair or not when receiving the range query. If it doesn't have the requested data, then it forwards to the neighboring nodes until the nodes which find the required key and if found send it back. To determine which node holds the (key, value) pairs, mapping method is required. Mapping method of peer node to DHT is called consistent hash function.

In the proposed system, labels of peers which hold keys of a given range query have generated and only need to map the peer node label to DHT. Here, the proposed indexing scheme is running over Chord DHT. The Chord DHT algorithm has used "SHA-1" hash function as a consistent hash function [12]. Nodes in Chord are place on Ring. Both node IDs and keys (hash from key-value pairs) are placed on the same ring. Secure Hash Algorithm (SHA-1) generates nodes' identifiers. SHA-1 hash produces a 160 bit digest from any data with a maximum of 2^{64} bits.

5.3. Proposed Indexing Scheme

In this system, the data set is postal addresses of [10]. So only need to consider the given range query of (R) issued by one peer node. The requested range (R) is received by the other peer node. This peer node is called initiator. This initiator firstly computes the labels of peers covered the range (R). In this phase, the initiator uses proposed indexing scheme shown in figure 3. The initiator node firstly traverses from the

root and compares the R with internal node's stored splitting point range. If R is covered with this internal or branch node, the labels of peer nodes can find by using the label of branch node. In other words, if R is covered by one branch node, all of the leaves of this branch node are the peer nodes which cover the requested range query (R). If not, the initiator continues the process of traversing and comparing through root to node. Here noticed that we need to traverse down step2 from root in left and right sub tree to start comparing. This is because pair of splitting point of x and y are only stored at that branch nodes of step2. But it doesn't need to traverse all nodes of the kd-tree. This is because of the efficient strategy of proposed indexing scheme. When the initiator traversing through local kd-tree, if any branch node's split interval is not covered R, the branch nodes and all of its child are pruned. So the initiator doesn't need to go down all of the child nodes. As a result, the proposed indexing scheme can reduce time consuming by pruning irrelevant nodes while query processing. And also the proposed algorithm can provide only exact peer labels so it can reduce the number of DHT lookups and thus can reduce bandwidth and latency consuming. In algorithm in figure 3, lv(pt) is denoted as stored point at the left subtree of root v, rv(pt) is the right subtree of v.

Algorithm: range_query Indexing(v, Range r)
Input: range query R, root of kd-tree
Output: labels of leaves (peers)
1. if v is a leaf then return label of v
2.else if (r $\sqsubseteq$ lv(pt)) then return the labels of lv and all its child nodes
3. else if (r $\sqsubseteq$ rv(pt)) then return the labels of rv and all of its child nodes

Figure 3. Range-query indexing algorithm

An example: Suppose that the queried range is bounded in range as [0.3, 0.8] on x axis and [0.4, 0.7] on y axis. Here kd-tree from figure 2 is used. The given range query is in rectangle shape and need to consider four points of [0.3, 0.4], [0.3, 0.7], [0.8, 0.4], [0.8, 0.7]. Firstly the initiator starts from root node. Then the initiator go down to the branch nodes at step 2 and starts the comparing the branch nodes with range R. Then labels of peers are resulted as #001, #010, #011, #11, and #1001. Then only need to perform DHT lookup using this labels.

6. System Architecture

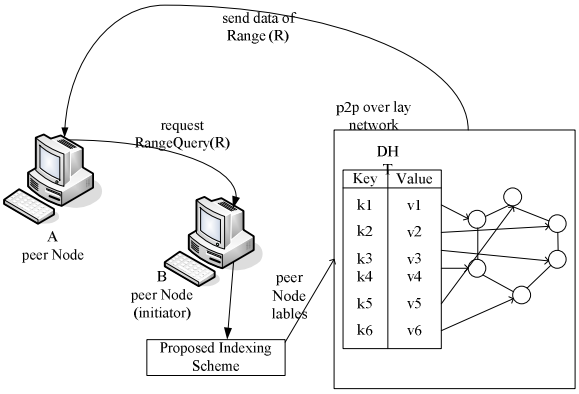


Figure 4. System architecture

In figure 4, system architecture is described. Firstly one of the peer node issues query request as R to the other peer node. This peer node starts the process as initiator. It uses it local kd-trees and locally computing the labels of relevant peer nodes using the proposed indexing scheme. When it gets the labels of peers, then it starts DHT lookup operation. If it finds all the (key, value) pairs, return all the data keys. Here DHT keys are the identifiers of peer nodes. If the

initiator node cannot find, it forwards the labels to its neighboring nodes and start the DHT lookup operation at these nodes. This process is continues until the keys are found.

7. Conclusion

In this paper, the efficient indexing scheme over DHT is proposed. By computing the identifier of peer nodes which can cover the given range queries before DHT lookup operation is performed, we can reduce number of DHT lookups. So the proposed system can also reduce the bandwidth and latency used in range query processing. To evaluate with other systems will be as the future work.

References

[1] S. Ramabhadran, S. Ratnasamy, J. M. Hellerstein, S. Shenker, "Prefix Hash Tree: An Indexing Data Structure over Distributed Hash Tables", *PODC*, 2004.

[2] C. Zheng, G. Shen, S. Li, S. Shenker "Distributed Segment Tree: Support of range query and cover query over DHT", in the *5th International Workshop on Peer-to-Peer Systems (IPTPS)*, Feb 2006.

[3] Y. Tang, J. Xu, S. Zhou, W. Chien Lee, "m-LIGHT Indexing: multidimensional data over DHTs", *29th IEEE International Conference on Distributed Computing Systems*, 2009.

[4] I. Stoica, R. Morris, D. Karger, M. F. Kasshoek, H. Balakrishnan, "Chord: A scalable Peer-to-Peer Lookup Service for Internet Applications", *in Proceedings of ACM SIGCOMM'01*, San Diego, September 2001.

[5] A. Rowstron, P. Druschel, "Pastry: A Scalable, decentralized object location and routing for large-scale peer-to-peer systems", in *Proceeding of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, Heidelberg, Germany, November 2001.

[6] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, "Tapestry: A Resilient Global-scale over for Service Deployment", *IEEE Journal On Selected Areas in Communications, Vol.22, No.1*, January 2004.

[7] S. Rantnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, " A Scalable Content-Addressable Network", *SIGCOMM'01*, San Diego, California, USA, August 27-31, 2001.

[8] X. Wei, K. Sezaki, "DHR Trees- A Distributed Multidimensional Indexing Structure for P2P Systems", *Scalable Computing: Practice and Experience, Volume 8*, November 3 2007, pp-291.

[9] A. W. Moore, "An inductor tutorial on kd-trees", *Technical Report No. 209, C*omputer Laboratory, University of Cambridge, 1991.

[10] http://www.rtreeportal.org/datasets/spatial/US/NE.zip

[11] H. M. Kakde, Range Searching using Kd Tree, 2005

[12] S. Sarmady, "A peer-to-peer Dictionary Using Chord DHT", *Report*, School of Computer Science, University Sains Malaysia, 2007.