Tree-based Indexing for DHT-based P2P Systems

Yi Yi Mar
University of Computer Studies, Yangon, Myanmar

Aung Htein Maw
University of Computer Studies, Yangon, Myanmar

Khine Moe Nwe
University of Computer Studies, Yangon, Myanmar

ABSTRACT

Nowadays, DHT-based P2P technology is used as a basis in many wide spread applications because of its scalability, robustness, and load balance. Many applications, including file sharing, communication and live video streaming are in a large distributed network environment. For an efficient and effective search in large data repositories, complex query processing becomes a major issue for DHT. Towards the goal of supporting complex queries in DHT-based P2P systems, this paper focuses on the usage of k-dimensional tree to build a tree-based index. The proposed index is built without modifying the structure of the overlay network. In this paper, the load balancing among peers is also considered according to the usage of kd-tree. Therefore the performance of kd-tree is studied and show that how it can affect the proposed index over P2P network. In this paper, PlanetSim simulator is used to implement the proposed index and evaluate the performance of the index by using various metrics.

Keywords

Indexing over DHT, DHT-based indexing system, Query processing over DHT, Indexing in structured P2P systems

1. INTRODUCTION

P2P system is a distributed system which facilitates the direct exchange of information and services between individual peers rather than relying on a centralized server. P2P forms the basis of many distributed computer systems, permitting each peer node to act as both a client and a server, consuming services from other available peers, whilst providing its own service to the rest of the network. P2P offers many advantages. These include scalability, high resource availability, no need for a centralized authority, and robustness. Peer-to-peer (P2P) technology is an increasingly popular vehicle for highly fault-tolerant, lightweight, and low-cost distributed computing environments. Cloud computing [1], digital music exchange [2], digital libraries [3], communication (e.g., Skype [4]), live video streaming (e.g., Zattoo [5]), secure data management systems [6], and bioinformatics databases [7] are just a few of the venues where this technology is being used today.

Since the late 1990, there have been P2P systems for organizing distributed systems in a way that there is no need for a global authority required to run the system. When the networks grew larger, Gnutella [8], one of the early P2P systems became inefficient in query processing because of message flooding through network. As a result, starting in 2001, distributed hash tables (DHT) such as Chord [9], CAN [10], Pastry [11], Tapestry [12], and Kademlia [13], were published. DHTs brought a great hype to peer-to-peer systems as they allow for large networks working autonomously using normal Internet connections.

DHT is actually a distributed data structure for storing of key and values pairs. DHT allows fast locating of data and can support exact match lookup when a key is given. For example, a DHT-based P2P system can use the exact match query interface with the file name as the key to publish and lookup a file. With the increase in the number of computers connected to the Internet and the emergence of a range of mobile computational devices which are equipped with mobile IP technology, the Internet is converging to a more dynamic, huge, extremely heterogeneous network. For the purpose of information dissemination and file sharing, P2P data management technology is being used in this large distributed environment. As a consequence, complex query processing may be a challenge for DHT-based P2P systems. For the purpose of supporting complex query processing over DHTs, a multi-dimensional index is needed to be built.

In this paper, k-dimensional tree or kd-tree to build an indexing scheme over structure P2P systems. The proposed tree-based index needs to consider two facts. The first is to keep resource sharing among peers balanced and the other is to be able to support the complex query (multi-dimensional and/or range query) in DHT-based P2P systems.

The rest of the paper is organized as follows. In Section 2, the existing indexing approaches over DHT-based P2P systems are described. And then Section 3 describes the architecture of proposed indexing scheme. The over view of kd-tree is discussed in Section 4. In Section 5, the proposed tree-based index is mentioned and the required steps to build this indexing scheme are described. In this Section, the process flow of of complex query by using the proposed index is also described. Then the performance of the proposed index is evaluated in Section 6. And then the summarization about the proposed system is mentioned in Section 7.

2. RELATED WORK

Complex query such as range query or multi-dimensional query are needed in many distributed applications, including content distribution, locality aware services, and resource discovery services such as file sharing applications. More and more applications require P2P systems to support complex queries over multi-dimensional data. For example, a P2P auction network [14] for real estate frequently needs to answer queries such as ‘select five available buildings closest to the airport’.

In any DHT-based P2P system, resources or data are distributed among peers by using keys of data. In a real DHT system, key of data may be keyword or one attribute of data value. For example, in a file sharing application, file name or keyword of file is defined as key of file. These keys are one-dimensional. Therefore DHT is very efficient in keyword query or exact match query. For the purpose of processing complex query over DHT, keys of data must be multi-dimensional

There are many approaches where data structures are fused with DHT to support complex queries [15]. Prefix hash tree [PHT] [16] is a distributed data structure that enables more sophisticated query over DHT. For efficiently processing one-dimensional query over DHT, it implemented trie-based distributed data structure. 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 overloading the root when the range is small.

Mercury [17] adopts a ring structure. Data are mapped to peers by their values, in order that the range query can be answered by the peers in a continuous region of the ring. To support multi-attribute queries, Mercury creates an individual ring for each attribute. A multi-attribute query is first processed based on the single arbitrarily chosen attribute in the corresponding ring, and then the other attributes are used as filter for retrieving final results. Load balance is guaranteed by periodically migrating peers form lightly loaded areas to heavily loaded areas in the ring. The major drawback of Mercury is the expensive cost to maintain multiple ring structures.

P-tree index structure is used to support equality and range queries in a distributed environment [18]. P-trees are highly distributed, fault-tolerant, and scale to a large number of peers. P-tree index ensures three properties, such as (1) it keep each peer at most between d and 2d entries (2) it limit each peer's storage space to be O (d. $\log_d$N), (3) it can indeed indexed the search key and ensures that no values are missed by the index structure. This index used Chord as an underlying ring structure. For consistency purpose, this index use Ping Process to detect the inconsistencies and repaired by the Stabilization Process. The key idea is to maintain parts of semi-independent B+- trees at each peer. Therefore it can be considered that it can only handle one-dimensional range query.

Squid, proposed by Schmidt et al [19], supports multidimensional range queries. They map the *n*-dimensional data to *1*-dimensioal space using Hilbert Space Filling Curves (HSFC). The one-dimensional data is then mapped to nodes arranged in a linear fashion along the NodeId ring. While this mapping allows range queries to be performed efficiently, the scheme loses load balancing property inherent to the DHTs. The authors proposed two load-balancing schemes (a) Load balancing at join time and (b) Load balancing at run time. The first scheme incurs an $O (N \log^2 N)$ communication cost and hence is very expensive of increase number of DHT pointers to maintain at each nodes. A node hosting *k* virtual nodes needs to maintain connections to $O (k \log N)$ DHT neighbours; hence, the scheme is not scalable with the number of documents in the system.

In [20], Ganesan et al. presented two P2P systems SCRAP and MURK. In the SCRAP, similar to Squid, the universe is first mapped down into a one dimension using a HSFC. The 1-dimension data is then range partitioned in one dimension and mapped onto Chord overlay network. The drawback of using HSFC in this case is that since when the dimension is high, the data which are near in multi dimensions will be far apart in one dimension and the locality property may not be guaranteed well. In MURK, partitioning is based on KD-Tree and is very similar to CAN. The MURK offers good data locality. However, some issues such as non-uniform number of neighbours and dynamic data distribution still need further study.

Distributed Hilbert R-trees (DHR-trees) [21] provides range query processing structure for P2P systems. It also makes use of HSFC in mapping from k-dimensional to 1-dimensional space. But similarly as in HR-Trees, this mapping is used only for ordering of each peer and upper regions in the tree, while the overlapping regions technique is inherited to facilitate multidimensional spatial queries in a more natural way.

Filling internal node with data can violate traversing down to leaf node. So Distributed Segment Tree [DST] [22] 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. However, it may lead maintenance overhead as key are replicated over internal nodes and leave nodes.

In [23], LIGHT can achieve efficient range query result. It proposed three mechanisms to construct the indexing structure over DHTs. LIGHT is high efficient in query processing but it still have drawback of bandwidth and latency. This drawback is based on the number of lookup operation. If we make more lookups on DHT, the more the bandwidth and latency is consuming.

There are many indexing systems built over Chord. But their network topology depends on the structure of data structure that they used. In this paper, a tree-based index is also built over Chord without modifying any changes in the structure of overlay network. This tree-based index is also a multi-dimensional index to support complex query processing over DHT-based P2P systems. It is also in a distributed manner.

3. SYSTEM ARCHITECTURE OF PROPOSED TREE-BASED INDEX

In this paper Chord is used as an overlay network. The proposed tree-based index is built over Chord to provide complex query processing over Chord. The architecture of the proposed index system is designed with a three-tier model as shown in figure 1. There are three layers in the DHT-based indexing system. They are (1) application layer (2) indexing layer over DHT and (3) local storage layer.

At the application layer, as shown in figure 1, each peer interfaces via one application system. User can send their desired query to the application interface and then receive the reply from this interface. Actually, when a query is requested, then the query is sent to a peer in the network from this application layer. When a peer receives a query, it starts searching the data.

At the indexing layer, the searching process is handled by the indexing scheme at the peer of overlay network. The indexing scheme at the peer starts to search data at its local storage.

At the storage layer, content data such as file, music (mp3) file, video, and application related information are stored. Peers handle searching of data in their local storage via indexing layer. If the data is not found in the peer's own storage, then it forwards the query to other peers in the network.

According to this architecture, the search process is only handled at the indexing layer. The searching or indexing scheme is efficient if it is clearly desirable to find the desired data in a minimal number of interactions with the system and the information returned by the system should be as concise and relevant as possible. The indexing process should also be simple and can handle user desired query type (complex query).

In this paper, the major work is to focus on the indexing layer to support the efficient searching process for complex query. DHT is very efficient in key word search or exact match query because keys of data are stored at the peers with the same or close IDs in the identifier space. The proposed system uses this lookup efficiency of DHT to get efficient lookup operation for complex query processing. Therefore it is important to define keys of data to be multi-dimensional in order to build multi-dimensional indexing scheme for storing and retrieving multi-dimensional data (complex query).

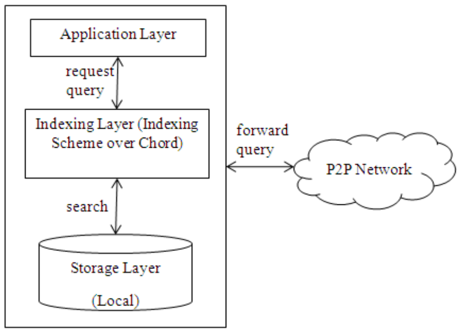
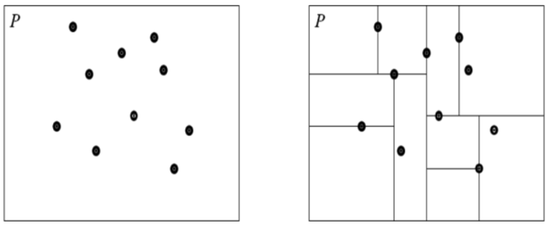


Figure 1. Three-tier architecture of proposed index system

For the purpose of efficiently handling users' desired complex queries, kd-tree is used to build a multi-dimensional tree-based index over Chord DHT. The main contributions are (1) labelling of data to generate multi-dimensional keys for storing data at peers (2) indexing mechanism at a peer. In this paper, the real dataset, DBLP [24] is used to test the performance of the proposed index.

4. KD-TREE OVERVIEW

A k-dimensional tree or kd-tree is a generalization of the simple binary tree used for sorting and searching [25]. It is a data structure for storing objects in k-dimensional space. It can be used for range search, nearest neighbour search and space partitioning. The basic intuition behind the kd-tree is to partition the data space by half-planes such that each point is contained in its own box-shaped region. The whole data space is decomposed into a relatively small number of cells such that no cell contains too many input objects. This provides a fast way to access any input object by position. Figure 1 shows a kd-tree construction for data point space ***P***, where each point in ***P*** has a set of dimensions {d_1, d_2, d_3... d_n}.



(a) (b)

Figure 2. Kd-tree construction for data space *P* (a) Input data point space: *P* (b) Output of *P*

Figure 2(a) shows the whole data space ***P***. And then kd-tree is built for P by using multi dimensions {d_1, d_2, d_3... d_n}. Figure 2 (b) shows the cell region of ***P*** after kd-tree has been built. The tree cycles through each dimension with increasing level number (i.e. level 0 splits the tree on dimension 'd_1', level 1 splits the tree on dimension 'd_2', etc.). In kd-tree, the data points are split into two subsets of roughly equal size [26]. While original kd-tree stored data in both internal and external nodes, the optimized kd-tree stores data only in the leaves of the tree. A drawback of kd-tree is that the structure is dependent on the order of insertion; deletion will also cause reorganization of the tree. The worse of all, since the division of hyper planes are defined by the position of points; the kd-tree may be highly imbalanced. An adaptive solution is to divide them to two subgroups with equal amounts of points [27].

In this paper, the balanced kd-tree is constructed by using optimal splitting threshold value, defined as $\mathbf{T_{SP}}$. $\mathbf{T_{sp}}$ is the maximum size of data on leaves of kd-tree. Therefore $\mathbf{T_{SP}}$ needs to be considered to be optimal. To define the optimal value for $\mathbf{T_{SP}}$ value, it is tested by assigning the values in the range of 100 to 1000. And then this paper evaluates the optimal $\mathbf{T_{SP}}$ value by using two metrics: (1) number of empty nodes in kd-tree and (2) number of peers with data size zero.

5. TREE-BASED INDEX OVER CHORD

5.1 Data Labelling

In DHT systems, keys of data are represented by using one dimension. In this proposed system, keys of data are multi-dimensional. To generate multi-dimensional keys for data, a balanced kd-tree is used. Initially the whole dataset is partitioned over kd-tree. Data are only stored at leaves of the tree. Labels of leaves in kd-tree are used as keys of data in these leaves. These data labels are multi-dimensional. These keys are used for locating data among peers of overlay network and searching data for user requested desired query from network. As shown in figure 3, there are two initial steps to build the proposed tree-based index. They are (1) data partitioning and (2) mapping data to peers.

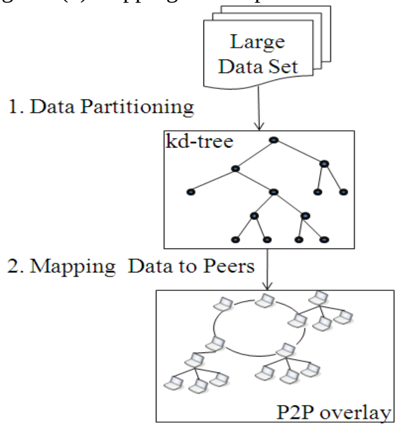
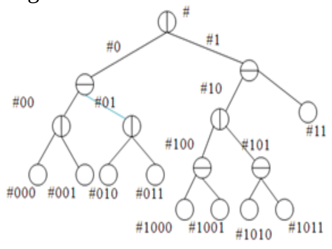


Figure 3. Two initial steps of proposed index

5.1.1 Data Partitioning

In this first step, keys of data are labelled by partitioning the large dataset on the kd-tree.



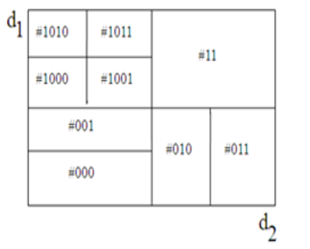


Figure 4. Kd-tree with two dimensions

Figure 5. Two-dimensional data cell region

As shown in figure 4, a two-dimensional kd-tree is built according to $\mathbf{T_{SP}}$. Half points of each dimension are generated while building kd-tree. These half point are also stored in tree-info list (**TIF**). In this system, each peer in overlay network needs to keep **TIF**.Data are only stored in leaves of the tree. Labels of data are defined while building kd-tree. Root of the tree is labelled with '#'. The label of left child is the concatenation of its parent's label with '0'and the right child has the label by concatenating its parent's label with '1'.In figure 5, leaves of kd-tree ae represented in the form of rectangular cell regions. Each label of cell region is key of the set of data in this cell. In this paper, leaves of kd-tree with data are values and data labels are keys.

When the kd-tree has been built, the next step is to generate data IDs for the purpose of storing data from kd-tree to peers in the Chord overlay network. In this paper, keys or labels of data are hashed by using a consistent hashing, SHA-1 [28] to generate data IDs.

5.1.2 Mapping Data to Peers

This section discusses how to store the data of kd-tree among peers in the network. In this paper, the proposed indexing scheme is built over an underlying DHT overlay network, Chord. It is a well known DHT. It uses keys to store and retrieve data. . In Chord, peers' identifiers are organized in a ring topology. In order to distribute data among peers, data needs to be mapped with the peers in a balanced manner. Load balancing is an issue in any P2P system. Resource sharing among peers needs to be balanced.

Random choice can provide balanced distribution [29]. So the set of keys and peers are required to be randomly chosen. For this purpose, standard hash function can be used to distribute data keys and peers' IPs hashed in randomness. Data IDs and peer IDs are computed by using SHA-1. In this paper, peer IDs are computed by hashing the IP address and data IDs by hashing of data keys. Then data IDs are distributed to the peers whose IDs are closest (less than or equal) to the IDs of peers.

In this paper, the system also considers load on each node is balanced while each peer has the load no more than $\mathbf{T_{pl}}$, where $\mathbf{T_{pl}}$ is the maximum load on each node. If most of peers in the network hold $\mathbf{T_{pl}}$, the P2P system will be balanced. $\mathbf{T_{pl}}$ can be computed according to equation (1), where $\mathbf{T_r}$ is the total amount of data in the system and N is the total number of peers in network.

$$\mathbf{T_{pl} = T_r / N} \qquad (1)$$

5.2 Indexing Mechanism

The indexing mechanism at a peer is shown in figure 6. After distributing data from kd-tree among in Chord overlay network, any peer can request a complex query for their desired data. This is performed at the first layer of three-tier model. When a peer receives a requested query via application interfaces, this peer starts searching process at the indexing layer. Most of the DHT-based indexing systems forward the query to the other peers in overlay network when the requested desired is not found in their local storage.

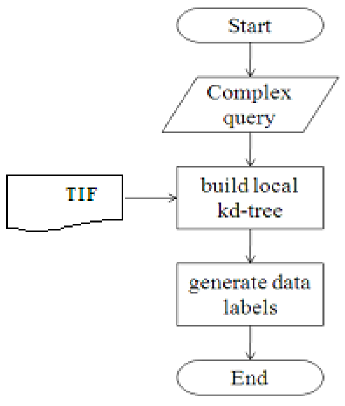


Figure 6. Indexing mechanism at a peer for one complex query

In most DHTs, key of requested data is the queries itself, i.e., file name of a file or author name of a book which is searched. Therefore they only handle one-dimensional query. In our indexing scheme, before starting searching process, multi-dimensional data keys are generated for a requested complex query by building the local kd-tree. The local kd-tree is built by using the half points stored at **TIF**. And then the peer checks which nodes of this local kd-tree the requested query can exist. If the peer found the nodes which can cover the requested query, labels of these nodes are used as keys of data for searching process at the indexing layer. This searching or lookup operation is the same as the efficient DHT lookup operation. Therefore our tree-based index is as efficient as the DHTs' efficient lookup and can also handle the complex query DHTs cannot handle.

6. EXPERIMENTAL SETUP

The performance of the proposed indexing scheme is evaluated via simulation. In this paper, we use PlanetSim simulator [30]. This is a java based simulator. It can easily be extended as the developer's desires and allows simulations to run on a variety of platform. In our performance evaluation section, there are two phase: (1) performance of kd-tree and (2) performance of the proposed index.

6.1 Evaluation of Kd-tree

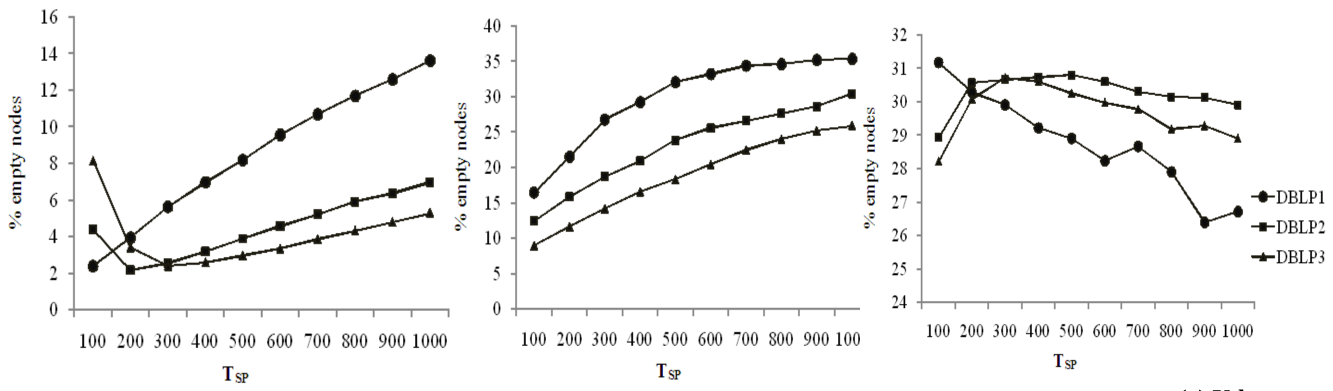
The proposed index is built based on the kd-tree. Therefore the performance of kd-tree is firstly evaluated to show how it can affect on the proposed indexing scheme. Performance of kd-tree is based on the value of $\mathbf{T_{SP}}$. According to $\mathbf{T_{SP}}$ kd-tree can be balanced or imbalanced. To keep kd-tree balanced, the value of $\mathbf{T_{SP}}$ needs to be optimal. $\mathbf{T_{SP}}$ is considered based on the following factors: (1) number of empty nodes on kd-tree (2) number of peers with data size zero and (3) number of wrong labels generated from the index. To define the optimal value for $\mathbf{T_{SP}}$, the required parameters are shown in table 1.

Table 1. Parameters for simulation

number of peers in overlay network	1000
range of $\mathbf{T_{sp}}$ values	100 to 1000
dimensions	2 dimensions (2D), 3 dimensions (3D) and 4 dimensions (4D)
size of DBLP dataset - (1) DBLP1 (2) DBLP2 (3) DBLP3	(1) 200 000 records (2) 500 000 records (3) 700 000 records

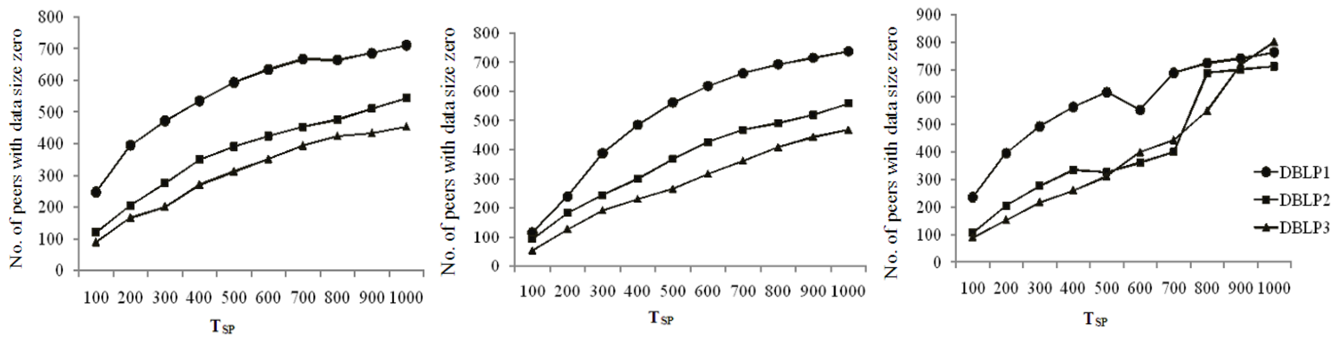
Figure 7 shows the percentage of empty nodes on kd-tree by using three dataset sizes DBLP1, DBLP2 and DBLP3. As shown in figure 7(a), the percentage of empty nodes is the least when $\mathbf{T_{SP}}$ is 200 and dataset is DBLP2. In DBLP1, 100 is the optimal $\mathbf{T_{SP}}$ value. In DBLP3, 300 is the optimal value. In figure 7(b), $\mathbf{T_{SP}}$ is optimal at 100 in all dataset sizes. Figure 7(c) shows that $\mathbf{T_{SP}}$ value is stable at the value of 200.

As shown in figure 8, $\mathbf{T_{SP}}$ is evaluated by showing how it can affect the load balancing among peers. Load balancing is an important issue in P2P system. One important point is that any indexing scheme in a P2P system should keep load balancing among peers. While mapping data among peers, some of peer nodes do not have data. The higher number of peers with data size zero can unbalance the underlying P2P network. Figure 8(a), (b) and (c) show that the greater the value of $\mathbf{T_{SP}}$, the higher the percentage of empty peer nodes. For this case, the value of $\mathbf{T_{SP}}$ is optimal at the value100.



(a) Kd-tree with 2D **(b) Kd-tree with 3D** **(c) Kd-tree with 4D**

Figure 7. Percentage of empty nodes on kd-tree



(a) Kd-tree with 2D **(b) Kd-tree with 3D** **(c) Kd-tree with 4D**

Figure 8. Percentage of peers with data size zero

The number of wrong labels is the most important factor in this proposed indexing system. Table 2 and table 3 show that how $\mathbf{T_{SP}}$ can affect the generating of wrong labels while kd-tree is built with 2D and 3D. When using kd-tree with 4 D, the indexing system does not produce wrong labels. Based upon our simulation results, we find that the higher the dimensions in a query, the lesser the number of wrong labels generated from our indexing scheme. As shown in table 2 and table 3, there are more wrong labels when $\mathbf{T_{SP}}$ value at 100 than $\mathbf{T_{SP}}$ value at 200.

Table 2. No. of wrong labels in 2D

T_{SP}	DBLP1	DBLP2	DBLP3
100	1	3	4
200	0	1	1
300	0	1	1
400	0	1	1
500	0	0	0
600	0	0	0
700	0	0	0
800	0	0	0
900	0	0	0
1000	0	0	0

Table 3. No. of wrong labels in 3D

T_{SP}	DBLP1	DBLP2	DBLP3
100	1	1	1
200	0	1	1
300	0	1	1
400	0	0	1
500	0	0	1
600	0	0	0
700	0	0	0
800	0	0	0
900	0	0	0
1000	0	0	0

According the above figure 7, figure 8, table 2, and table 3, $\mathbf{T_{SP}}$ is optimal at 200. With this optimal $\mathbf{T_{SP}}$, kd-tree is balanced and number of peers with data size zero is minimum than other $\mathbf{T_{SP}}$ values. The number of wrong labels can increase the lookup overhead and response time in the indexing scheme. With $\mathbf{T_{SP}}$ at 200, the number of wrong labels is the least according to table 2 and table 3. Therefore $\mathbf{T_{SP}}$ value at 200 is used to build kd-tree in the proposed tree-based index.

6.2 Evaluation of Tree-based Index over Chord

After evaluating the performance of kd-tree, the kd-tree is balanced at $\mathbf{T_{SP}}$ with 200. And then this balanced kd-tree is used to build multi-dimensional indexing scheme over Chord. So we need to consider the performance of the proposed tree-based index for processing of complex query. To evaluate the performance of the tree-based index, four metrics are used to measure. They are (1) key distribution (2) number of lookup hops, (3) lookup message overhead and (4) response time. In this evaluation, the proposed indexing scheme is tested with the number of peers up to 10000 and four-dimensional range query. For this experiment, 30 DHT lookups are conducted for a complex query. The numbers of lookup hops in average, average lookup message overhead and average response time are shown in following sections.

6.2.1 6.2.1. Key Distribution

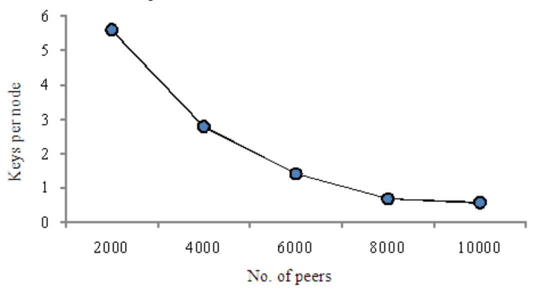


Figure 9. Keys per peer

Key distribution is determined with the distribution of keys among peers. This is measured as keys per peer. Also the total number of keys stored in the network is calculated according to equation (1) shown in Section 5.1.2. Figure 9 shows that the larger the network the less keys per peer.

6.2.2 Number of Lookup Hops

The number of routing hops required for lookups was measured by calculating the number of peers that must be visited until the peer with the requested information is found. The last hop added into the count is the one reaching the target node. Figure 10 shows that the number of lookup hops only slightly increase when the number of peers greater.

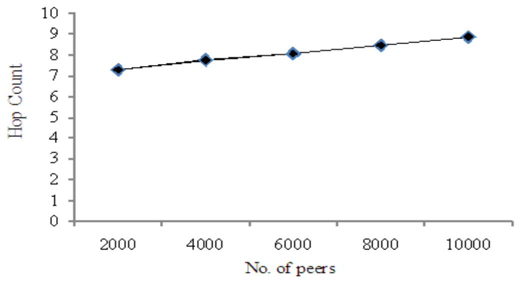


Figure 10. Hops per lookup

6.2.3 Lookup Message Overhead

This parameter determines the number of messages and the volume of data transferred in a lookup process. Lookup message overhead is calculated by multiplying the number of lookup hops by the association message sizes. In our simulation, we use the message size with 58 bytes. Figure 11 shows that message overhead only slightly increases when the network scales.

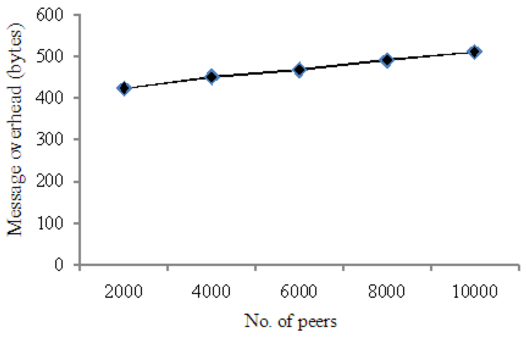


Figure 11. Message overhead per lookup

6.2.4 Response Time

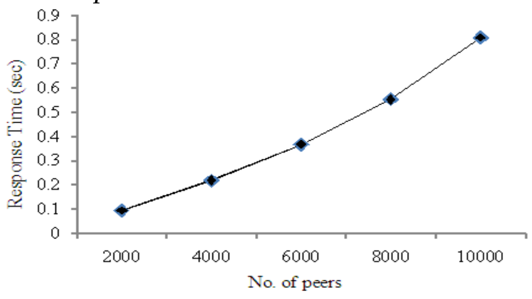


Figure 12. Response time for a requested query

The response time of a requested query is one of the most important factors in our simulations. In our simulation, the response time is the time starting from the lookup process to the end of the successfully receiving the requested query. It is measured in seconds. Figure 12 shows that the greater the numbers of peers in network the higher the response time.

According to the results shown in figure 10 and figure 11, the number of lookup hops and lookup message overhead only slightly increase the network scale. The index can achieve a lookup operation for a complex query with the number hops less than the maximum hops of a lookup in Chord, O (log N). As shown in figure 12, the response time is considerably increased. Based on our simulation results, we find that the proposed indexing can achieve complex query processing regardless of the growing of the network.

7. CONCLUSIONS

In this paper, the proposed tree-based indexing scheme is simulated with PlanetSim simulator. According to the simulated results based on the metrics of (1) number of empty nodes on kd-tree, (2) number of peers with data size zero, and (3) number of wrong labels, it is observed that $\mathbf{T_{SP}}$ is optimal at 200 and it can keep kd-tree balanced. When implementing the multi-dimensional index over Chord, there is no need to modify the structure overlay network. The evaluation results in Section 6.2 show that the proposed indexing scheme can handle complex query and also keeps data availability independent of the network scales. This is because the proposed index can achieve a complex query with the number of hops in O (log N). In the future step, the proposed indexing scheme will be proved by comparing with the existing approaches with the measurement of bandwidth and latency consuming. And then various types of query will be tested by using this tree-based index to show how it can effect over DHT-based P2P systems.

8. REFERENCES

[1] F. Marozzo, D. Talia, and P. Trunfio, “P2P-MapReduce: Parallel data processing in dynamic Cloud environment”, *Journal of Computer and System Sciences*, May 2012.

[2] P. J. Alexander, “Peer-to-Peer File Sharing: The Case of the Music Recording Industry”, *Review of Industrial Organization*, vol. 20, no. 2, pp. 151-161, March 1 2002.

[3] A. Amrou, K. Maly, and M. Zubair, “Freelib: Peer-to-peer-based Digital Libraries”, in *Proceedings of the 20th International Conference on Advanced Information Networking and Applications*, 2006.

[4] “Skype”. http://www.skype.com

[5] “Zattoo-Live TV and More”, http://zattoo.com.

[6] K. W. Hamlen and B. Thuraisingham, “ Secure Peer-to-peer Networks for Trusted Collaboration”, in *International Conference on Collaborative Computing: Networking, Applications and Worksharing*, November 2007.

[7] P. H. Chou, R. B. Ortega, and G. Borriello, “The Chinook Hardware/Software Co-synthesis System”, in *Proceedings of the 8th International Symposium on System Synthesis*, 1995.

[8] M. A. Jovanovic, F. S. Annexstein, and K. A. Berman, “ Scalability Issue in Large Peer-to-Peer Networks- A Case Study of Gnutella”, Cincinati Univ., Technical Report, 2001.

[9] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, “Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications”, *ACM SIGCOMM Conference*, 2001.

[10] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, “A Scalable Content-Addressable Network”, in *Proceedings on the Conference on Applications,*

technologies, architectures, and protocols for computer communications, USA, 2001.

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

[12] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A Resilient Global-Scale Overlay for Service Deployment", *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, January 2004.

[13] P. Maymounkov and D. Mazieres, " Kademlia: A Peer-to-peer Information System Based on the XOR Metric", in *Proceedings of The First International Workshop on Peer-to-Peer Systems*, London, 2002.

[14] E. Tanin, A. Harwood, and H. Samet, "A Distributed Quadtree Index for Peer-to-Peer Setting", in *Proceedings of the 21st International Conference on Data Engineering*, April 5-8, 2005, Tokyo, Japan.

[15] P. Yalagandula and J. Browne, "Solving Range Queries in Distributed System", Tech. Rep. TR-04-18, UT CS, 2003.

[16] S. Ramabhadran, S. Ratnasamy, J. M. Hellerstein, and S. Shenker, "Prefix Hash Tree: An Indexing Data Structure over Distributed Hash Tables", in *Proceedings of Conference on Applications, technologies, architectures, and protocols for computer communications*, USA, 2005.

[17] Y. Shu, B. C. Ooi, K. L. Tan, and A. Zhou, " Supporting Multi-dimensional Range Queries in Peer-to-Peer Systems", in *Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing*, USA, 2005.

[18] A. Crainiceanu, P. Linga, J. Gehrke, and J. Shanmugasundaram, " Querying Peer-to-Peer Networks Using P-Trees", *7th International Workshop on the Web and Databases*, Paris, France, June 17-18, 2004.

[19] C. Schmidt and M. Parashar, " Flexible Information Discovery in Decentralized Distributed Systems", in *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, 2003.

[20] P. Ganesan, B. Yang, and H. G. Molina, " One Tours to Rule them All: Multi-dimensional Queries in P2P Systems", *Seventh International Workshop on the Web and Databases*, Paris, France, June 17-18, 2004.

[21] X. Wei and K. Sezaki, "DHR-TREES: A Distributed Multi-dimensional Indexing Structure for P2P Systems", *Scalable Computing: Practice and Experience*, vol. 8, November 2007.

[22] C. Zhen, G. Shen, S. Li, and S. Shenker, "Distributed Segment Tree: Support of Range Query and Cover Query over DHT", in *Proceedings of the 5th International Workshop on Peer-to-Peer Systems*, February 2006.

[23] Y. Tang, S. Zhou, J. Xu, and W. C. Lee, " A Lightweight Multi-dimensional Index for Range Queries over DHTs", *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 12, pp. 2046-2054, December 2011.

[24] "DBLP". *http://dblp.uni-trier.de/xml*.

[25] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An Algorithm for Finding Best Matches in Logarithmic Expected Time", *ACM Transactions on Mathematical Software*, vol. 3, no. 3, September 1977.

[26] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, Springer, 2008

[27] M. Wu, Master Thesis-2006.pdf. On R-tree Index Structures and Nearest Neighbour Queries.

[28] FIPS. PUBS 180-2 Secure Hash Standard U.S. Department of Commerce/NIST, August 1, 2002.

[29] I. Stoica, R. Morris, D. Liben-Nowell, D. r. Karger, M. F. Kaashoek, F. Dabek, and H. Balarishnan, "Chord: AScalabel Peer-to-peer Lookup Protocol for Internet Applications", *IEEE/ACM Transactions on Networking*, vol. 11, pp. 17-32, February 2003.

[30] P. Garcia, C. Pairot, R. Monderjar, J. Pujol, H. Tejedor, and R. Rallo, "PlanetSim: A New Overlay Network Simulation Framework", in *Proceedings of the 4th International Conference on Software Engineering and Middleware*, 2005.