Usage of Kd-tree in DHT-based Indexing Scheme

Yi Yi Mar, Aung Htein Maw, and Khine Moe Nwe

***Abstract*—Distributed Hash Table (DHT) is a promising approach for a distributed data management platform in a large-scale network environment. In order to provide an efficient query processing and resource sharing in information discovery services, any DHT-based P2P system essentially needs to build an efficient indexing scheme for supporting users' desired queries (complex query, i.e., multi-dimensional and/or range query). In this paper, multi-dimensional indexing scheme is built by generating multi-dimensional keys. For this purpose, k-dimensional tree (kd-tree) is proposed to be used in the indexing scheme. This paper focuses on the performance of kd-tree regarding to its splitting threshold value and evaluates it by using three metrics: (1) number of empty nodes on tree (2) number peers with data size zero, and (3) number of wrong labels. According to the simulated results, this paper defines that kd-tree is built at the splitting threshold value "200" and builds a multi-dimensional indexing scheme over Chord. This paper shows that how the proposed indexing scheme with the usage of kd-tree supports the complex query processing over Chord.**

***Index Terms*—Indexing over DHT, Complex query processing over DHT, DHT-based P2P query processing**

I. Introduction

DHT technique is popular in many distributed applications including resource sharing, network monitoring services, and event notification services. In DHTs, data items are spread over networked computers such as Chord [1], Tapestry [2], Pastry [3] and CAN [4]. It allows fast locating of data and can support exact match lookup when a key is given. But it is only a one-dimensional indexing mechanism. For an efficient searching in information discovery service, complex query processing is a major challenge for DHTs.

In this paper, we use kd-tree to build a multi-dimensional index over DHT. Firstly, balanced kd-tree is built to partition the entire resources. Secondly resources on kd-tree are distributed among peers of overlay network. When a query is requested, our indexing scheme produces multi-dimensional keys for the desired query. As our indexing scheme is based on kd-tree, we focus on the performance of kd-tree and evaluate it to show how it can affect on the indexing scheme.

II. Related Works

A balance data structure can balance the distribution of resources among peers [5]. Therefore recent DHT-based indexing systems for handling complex query by using a data structure such tree, graph, and grid.

Prefix Hash Tree, PHT [6] is the first indexing scheme over DHT that is efficient for one-dimensional range query. It proposed two algorithms for range query processing. The first resulted in high latency because all leaves are sequentially traversed until the query is completely solved. In second one, query processing is done in parallel and the query is recursively forwarded until the leaf nodes are overlapping the query. When the requested range is small, it may lead overloading.

To solve the overhead in PHT, DST [7] fills the internal nodes with data to violate traversing down to leaf node. So it stores keys in both leaf nodes and internal nodes. To process a range query, firstly, this query is decomposed into a union of minimum node intervals of segment tree. Finally the query is solved by the union of keys returned from the corresponding DST nodes. It may lead maintenance overhead because keys are replicated over leaves and internal nodes.

DAST [8] is built for range query processing. It firstly constructs an arbitrary segment tree and encapsulate the (key, data) pairs with segmentIds. When processing range query, it divides the requested query into the segments as in AST (arbitrary segment tree). And then it retrieves the data related with segmentIds. DAST can reduce the number of DHT retrievals. But AOR (accuracy of range) can drop because the union of segmentIds can also contain the irrelevant segmentIds.

DHR-trees [9] provide range query processing structure for P2P systems. It can achieve efficient query processing. But it cannot handle multidimensional query with one index, whereas it reduces the m dimensions to one dimension.

In [10], m-LIGHT also used kd-tree to build an indexing scheme over DHT. It proposes a new data aware splitting strategy to distribute data on kd-tree. And then it also proposed a new mechanism to map data from kd-tree to peer nodes. It is high efficient in query processing but still has the drawback of bandwidth and latency consuming.

In the above approaches, complex query is provided by a single index which is created by replicating and combining of all attributes and a multiple-index created by combining the results of each attribute's index.

III. Indexing System with Kd-tree

The architecture of the proposed index system is designed with a three-tier model as shown in fig.1. The first layer is the application layer and it is for the interactions among peers. The second layer is the indexing layer over DHT which mainly handles the query processes. At the third layer or storage layer, content data such as files like mp3, video, and application related information are stored.

At the application layer, each peer interfaces through an application system. User can send their desired queries and receive the results via this application interface. When a query is requested, then the query is sent to one peer in the network from this application layer. When a peer receives a query, it starts searching the data at the indexing layer of this peer's

Manuescript received December 19, 201; revised January 27, 2013.

Yi Yi Mar is with the University Computer Studies, Yangon, Myanmar (e-mail:yyimar@gmail.com).

Aung Htein Maw is with the University of Computer Studies, Yangon, Myanmar (e-mail: aunghteinmaw@gmail.com).

Khine Moe Nwe is with the University of Computer Studies, Yangon, Myanmar (e-mail: kmnweucsy@gmail.com).

side. The searching process is handled by the proposed indexing scheme at this indexing layer. The indexing scheme at the peer starts to search data at its local storage. If the value is not found in the peer's own storage, then this peer 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 need to handle user desired queries.

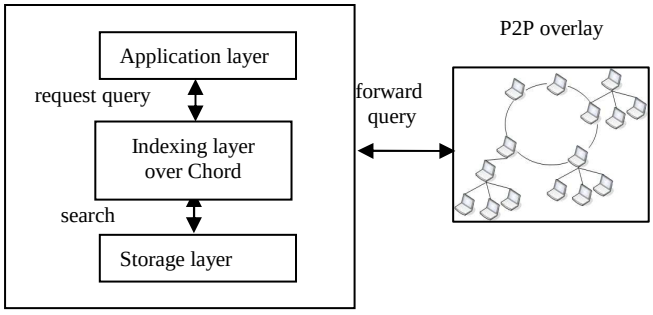


Fig.1. Three-tier architecture of proposed index system

In this paper, we 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 their identifier space. We use this lookup efficiency of DHT to get efficient lookup operation for complex query processing. Therefore we need to define keys of data to be multi-dimensional in order to build a multi-dimensional indexing scheme for storing and retrieving of data. The proposed indexing scheme is described in the following parts: (A) labeling of data to generate multi-dimensional keys (B) distributing data to peers (C) indexing mechanism. In this paper, the real dataset, DBLP [11] is used to test the performance of the proposed indexing scheme.

A. Data Labeling

In this paper, data labeling is the process of generating multi-dimensional keys for data. In most pure DHT systems, keys of data are represented by using one dimension. In this proposed system, keys of data are generated in the multi-dimensional forms. In this paper, kd-tree is used to generate multi-dimensional keys. K-d tree is a data structure extended from binary search tree of one-dimension to k-dimensions. It is very useful in several applications, such as multi-dimensional and range query processing [12], [13], geographic information systems and computer graphic systems. Fig. 2 shows the simple construction of kd-tree by using two dimensions namely as $\mathbf{d_1}$ and $\mathbf{d_2}$. It can be constructed by recursively partitioning the entire dataset evenly along each dimension in an alternative fashion. The partitioning process will be stopped as soon as the number of data in each tree node is no more than the predefined splitting threshold. In this paper, this splitting threshold is defined as $\mathbf{T_{SP}}$ which is the maximum loads of each node in kd-tree. A major drawback is that kd-tree may be highly imbalanced [14]. An adaptive solution is to divide the data to two subgroups with equal amounts of data [15]. So $\mathbf{T_{SP}}$ essentially needs to be considered to be an optimal value. $\mathbf{T_{SP}}$ value is defined via simulated results discussed in section V.

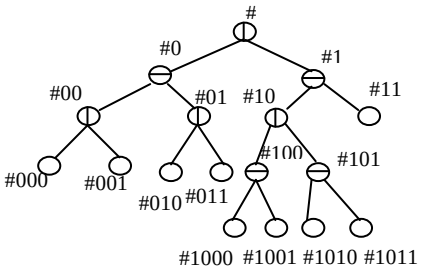


Fig.2. Data partitioning on kd-tree

In this paper, labels of data are defined while building kd-tree for the purpose of generating multi-dimensional keys for data. As shown in fig.2, label of root node is "#" and the label of branch and leaf nodes are the concatenation of its own label (0/1) and the label of parent's node label. The label of left child is label of parent's plus "0" and right child has the label of parent's plus "1".

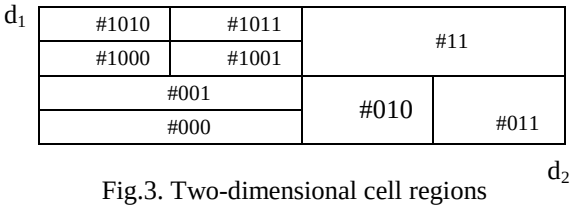


Fig.3. Two-dimensional cell regions

In fig.3, leaf nodes of kd-tree are represented in the form of rectangular cell regions. Each cell contains the related data (records) regarding with 2D partitioning. While building kd-tree, half points of each dimension are generated while building kd-tree. Data are only stored in leaf nodes of kd-tree. Data labels are keys and leaf nodes with data are values. In this proposed system, kd-tree info with half points of dimensions is recorded in tree info list **TIF**, and **TIF** is stored in each peer.

B. Distributing Data to Peers

The section shows how to distribute the data from leaves of kd –tree to peers. In this paper, the proposed indexing scheme is built over a DHT overlay network, Chord. In order to distribute data among peers, data keys need to be mapped among the peers. In any P2P, data sharing needs to be balanced. Random choice can provide balanced distribution [16]. For this purpose, standard hash function SHA-1 [17] is used. 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 peer IDs. In this paper, we consider 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 peer. $\mathbf{T_{pl}}$ is calculated by using (1), where $\mathbf{T_r}$ is the total amount of data in the system and **N** is the total number of peers in network.

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

C. Indexing Mechanism

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. They only handle one-dimensional query. A P2P auction network [18] for real estate frequently needs to

answer a complex query such as 'select five available buildings closest to the airport'. This query is multi-dimensional range query (complex query). In this proposed indexing mechanism at a peer is shown in fig.4.

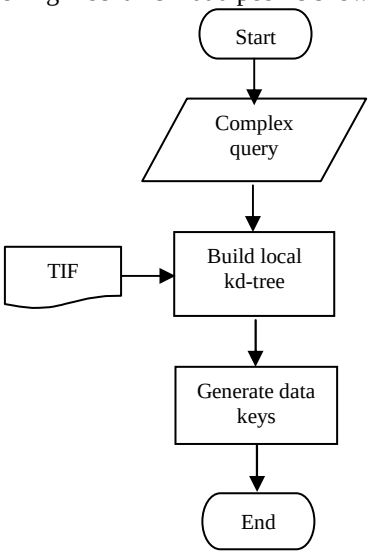


Fig.4. Indexing mechanism at a peer

When a complex query is requested, a peer starts searching process as an initiator. 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. This searching process is the same as the exact match DHT lookup operation. Therefore our indexing scheme is as efficient as the DHTs' efficient lookup and can also handle the complex query over Chord while DHTs cannot handle such query.

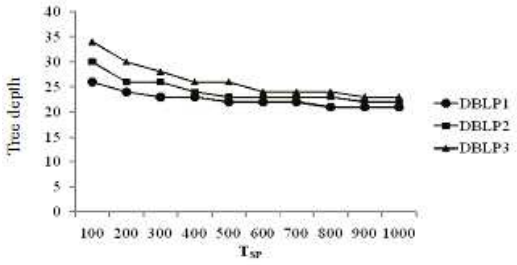
IV. Experimental Setup

The proposed indexing scheme is simulated by using Java language. Required parameters for simulation are shown in table I.

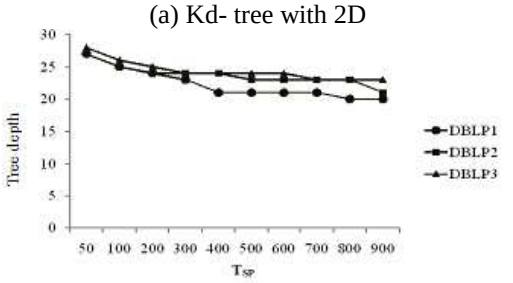
TABLE I. Parameters for simulation

number of peers	1000 peers
three sizes of DBLP dataset	-DBLP1 with 200 000 records -DBLP2 with 500 000 records -DBLP3 with 700 000 records
T_{SP}	1 to 1000
number of dimensions	multi dimensions

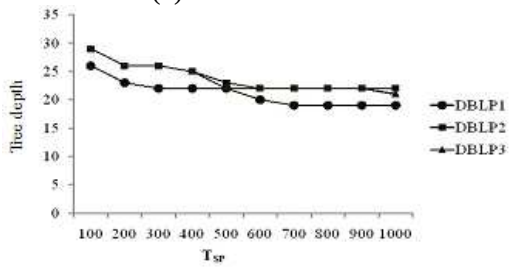
In this simulation, the system uses peers (network size) in the range of 1 to 1000. DBLP dataset is used as DBLP1, DBLP2, and DBLP3. We use three different dimensions such as 2 dimensions (2D), 3 dimensions (3D) and 4 dimensions (4D). The reason why we use up to 4D in this paper is according to the simulated results shown in table II and table III. T_{SP} value is also assigned with various values between 1 and 1000. Before the evaluating the performance of kd-tree, this paper firstly describes the tree depths of kd-tree and number of data keys generated from kd-tree are shown in fig.5 and fig.6.



(a) Kd- tree with 2D

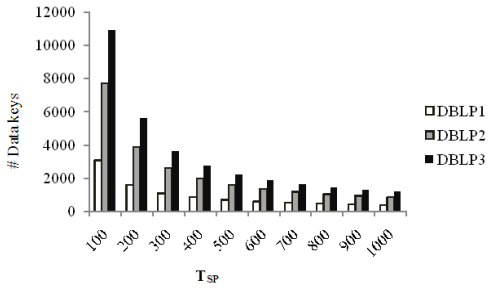


(b) Kd-tree with 3D

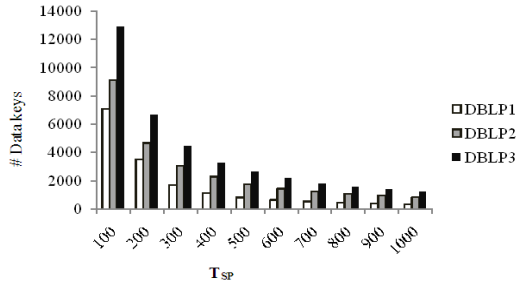


(c) Kd-tree with 4D

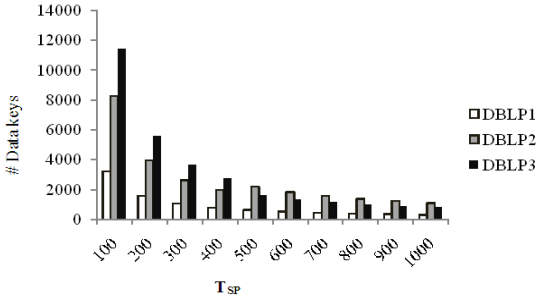
Fig.5. Leaf depths in kd-tree



(a) Kd-tree with 2D



(b) Kd-tree with 3D



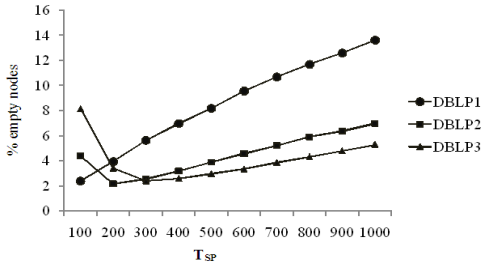
(c) Kd-tree with 4D

Fig.6. Key Generation

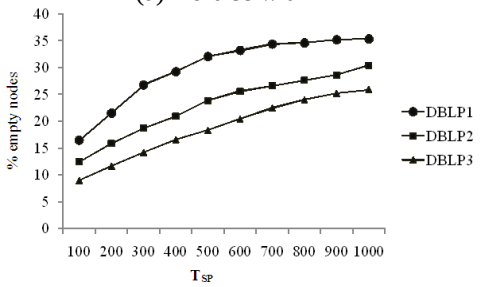
As shown in fig.5, tree depths are different when the value of $\mathbf{T_{SP}}$ varies between 1and 1000. Kd-tree is built by using 2D, 3D, and 4D as shown in fig.5 (a), (b), and (c). In all these figures, tree depths are getting lower when the value of $\mathbf{T_{SP}}$ is greater. Fig.6 shows the number of data keys which are obtained after kd-tree has been built with 2D, 3D, and 4D. The data keys are the labels of leaf nodes in kd-tree. From the simulation, it can be observed that the greater the value of $\mathbf{T_{SP}}$, the less number of data keys are generated.

V. Performance Evaluation of Kd-tree

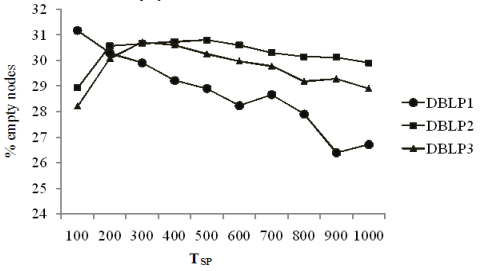
The performance of kd-tree is evaluated via simulations with three metrics (1) number of empty nodes on kd-tree (2) number peers with data size zero, and (3) number of wrong labels. After partitioning data on kd-tree, a large number of empty nodes can make it imbalance. To keep kd-tree balance, the optimal $\mathbf{T_{SP}}$ value is required to define.



(a) Kd-tree with 2D



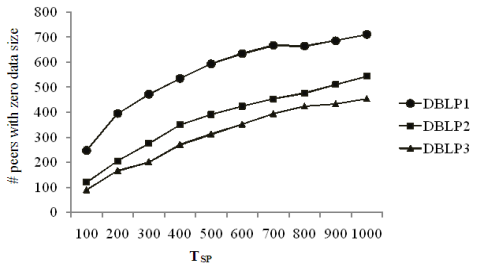
(b) Kd-tree with 3D



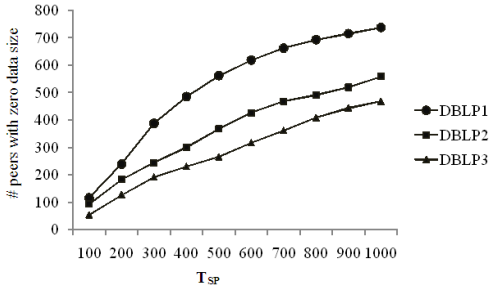
(c) Kd-tree with 4D

Fig.7. Percentage of empty nodes in kd-tree

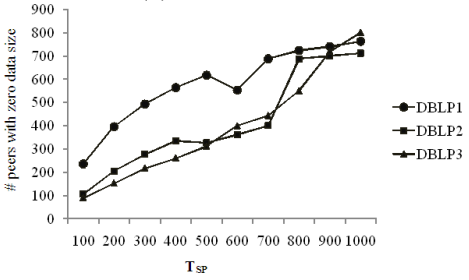
As a first step, $\mathbf{T_{SP}}$ is evaluated based on the number of empty nodes on kd-tree. Fig.7 shows the percentage of empty nodes on kd-tree using 2D, 3D, and 4D. As shown in fig.7 (a), the percentage of empty nodes is the least when $\mathbf{T_{SP}}$ is 200 and dataset is DBLP2. In DBLP1, 100 is the optimal $\mathbf{T_{SP}}$ value. In DBLP3, 300 is the optimal value. In fig. 7(b), $\mathbf{T_{SP}}$ is optimal at 100 in all dataset sizes. Fig.7(c) shows that $\mathbf{T_{SP}}$ value is stable at the value of 200. Secondly, $\mathbf{T_{SP}}$ 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. Fig.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

Fig.8. Percentage of peers with zero data size

TABLE II: # Wrong labels with 2D

T_{SP}	#Wrong label (DBLP1)	#Wrong label (DBLP2)	#Wrong label (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 III: # Wrong labels with 3D

T_{SP}	#Wrong label (DBLP1)	#Wrong label (DBLP2)	#Wrong label (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

Generating labels or data keys or data locations is an important step in our indexing scheme. According to the number of wrong labels, the performance of the proposed indexing system can become bad. Therefore the number of wrong labels is the most important factor in this proposed indexing system. Table II and table III show that how $\mathbf{T_{SP}}$ can affect the generating of wrong labels while kd-tree is built with 2 dimensions and 3 dimensions. This proposed indexing system can generate the various numbers of wrong labels according to the value of $\mathbf{T_{SP}}$. According to this results in $\mathbf{T_{SP}}$ value at 100 always generate wrong labels. When using kd-tree with 4 dimensions, the indexing system does not produce wrong labels. According the above fig.7, fig.8, table II, and table III, $\mathbf{T_{SP}}$ is optimal at the value "200".

VI. Conclusion

In this paper, kd-tree is proposed to use in the portion of data management to build a multi-dimensional indexing scheme over Chord. It can provide data with multi-dimensional keys for the purpose of storing and retrieving data to meet the user's desired queries. According to the simulated results, the proposed indexing scheme can handle multi-dimension by a single multi-dimensional indexing scheme by using kd-tree. This proposed indexing scheme can be applied in large data management applications, including file sharing, communication, and live video streaming.

References

[1] 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.

[2] 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.

[3] A. Rowstron and P. Druschel, " Pastry: 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*, Heidelberg, Germany, November 2001.

[4] 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.

[5] L. Lymberopoulos, S. Papavassiliou and V. Maglaris, "A Novel Load Balancing Mechanism for P2P Networking", in *GridNets*, Lyon, France, 2007.

[6] 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.

[7] 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.

[8] X. Chen and S. A. Jarvis, "Distributed Arbitrary Segment Trees: Providing Efficient Range Query Support over Public DHT Services", in *Proceeding of The 18th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, 2007.

[9] 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.

[10] Y. Tang, J. Xu, S. Zhou and W. Lee, "m-LIGHT: Indexing Multi-Dimensional Data over DHTs", in *Proceeding of 29th IEEE International Conference on Distributed Computing System*, 2009.

[11] DBLP. Available: http://dblp.uni-trier.de/xml

[12] F. P. Preparata and M. I. Shamos, *Computational Geometry- An Introduction*, Springer-Verlag, USA.

[13] A. W. Moore, "An Introductory Tutorial on Kd-Trees", Ph.D. dissertation, University of Cambridge, 1991.

[14] S. Sarmady, "A peer-to-peer Dictionary Using Chord DHT", University of Sains Malaysia, Technical Report, 2007.

[15] M. Wu, "On R-tree Index Structures and Nearest Neighbor Queries", MS Thesis, University of Houston, December, 2006.

[16] 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.

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

[18] 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.

Yi Yi Mar received the master degree in computer science (M.C.Sc) from University of Computer Studies, Yangon (UCSY), Myanmar, in 2008. She is currently working toward the PhD degree in UCSY. Her research interests include Internet computing, data management, peer-to-peer networks and distributed systems.

Aung Htein Maw received the Master of Information Science (M.I.Sc) degree from University of Computer Studies, Yangon (UCSY), in 2001, the master degree in Engineering Physics (Electronics) from Yangon Technological University (YTU), Myanmar, in 2002, and the PhD degree in Information Technology from UCSY, in 2009. He is a lecturer and head of department of Physics, UCSY. His research interests include wireless sensor network, virtualization technology and distributed computing environment. He has published technical papers in these areas, in the conference proceedings, including the International Conerence on GeoInformation Technology for Natural Disaster management and Rehabilitation, Bangkok, Thailand and International Conference on Computer Application, Yangon, Myanmar. He has been cooperated at ICTTI (Information and Communication Training Institute) in 2009 founded by JICA and UCSY as a network counterpart. He is also a member of operation committee of SOI Asia in UCSY site and cooperates in e-learning courses and workshop.

Khine Moe Nwe received the Master of Information Science (M.I.Sc) degree from University of Computer Studies, Yangon (UCSY), Myanmar, in 1998 and the PhD degree in Information Technology from UCSY in 2004. She is an associate professor in the department of Software, UCSY. She has published technical papers related with netwrok security in the proceedngs of International Conference on Computer Application, Yangon, Myanmar. Her research interests inclue network security, cloud computing, OS and distributed systems.