*2009 International Conference on Computer Engineering and Applications*
*IPCSIT vol.2 (2011) © (2011) IACSIT Press, Singapore*

# Keyword Search in P2P network Based on Trie Tree

Zhu Juan[1], Jin Deqiang[1], Mo Siquan[1]

[1] School of Computer Science & Engineering, South China University of Technology, Guangzhou

**Abstract.** In the past few years, keyword search is widely uses in the network. In the P2P networks it always uses an index table to implement. In this paper, we present a distributed index table can be performed efficiently to in locate the keyword.

**Keywords:** Peer-to-Peer (P2P), Keyword Search, Trie Tree, Distributed Hash Table

## 1. Introduction

In the structured P2P networks, a virtual DHT (Distributed Hash Table) is consisted of the nodes in the overlay. The data is to assigned to be stored in the specific node. Both the data and the nodes are mapped to a key (we called it DHTID) and a value. The DHTID is calculated through the consistent hash function. The value can be an address, a document, or an arbitrary data item. Examples of DHT networks include Chord [1], Can [3], Pastry [2], Tapestry [4].They locate the data with the difference route method.

Because the data is stored in the specific node in the DHT overlay, to locate a data should calculate the node firstly. Then jump to the objective node by the route table. So the data can be found the key's value in the node. However, this approach must be limited to supply the exact name of the data, so that the key calculated by the hash function can be found in the DHT overlay, and it can not support the keyword search.

In order to achieve the goal of keyword search, the most usual way is designing a specialized index table. Through this index table the set of DHTIDs can be found to the corresponding keyword. After that the real data can be located by the DHT algorithm. For instance, DataA includes the keywords Keyword1 and Keyword2, DataB includes the Keyword2. So there are two items in the index table, depicted in Table 1. Thus we can find the corresponding data by the keyword. There are many systems which have this function (e.g., [8], [9], [10], [11], [12]).

Table 1: keyword index

| keyword | data |
|---|---|
| Keyword1 | DataA |
| Keyword2 | DataA, DataB |

However, there are a lot of problems in the normal index table: First is the storage space of the index table. When there are less data in the network, the index table is also very small, and it will not take up too much storage space. But the more data is added into the network, the more keywords will be, and the storage space of index table can not be ignored.

The second issue is the maintenance of the index table. If you have stored a index table in each node in the network, once there is a change in the index table, all the other node needs to update the index table. So it suffered too many troubles to maintain this table.

+ Zhu Juan. Tel.: +8613450296504.
*E-mail address*:hatmann1944@gmail.com.

The third problem is the capability of fault-tolerant. Once the index table is error, the keyword search on this node will fail.

In addition, if the index table is stored distributed, to maintain more convienient and make each node of load more balanced, how to store and organize?

In order to solve the problems similar to those mentioned above effectively, in accordance with the basic thinking of the index table, we create a distributed index table by some of the nodes in the network. Each index node takes charge of a specific keyword. To look for a certain keyword can pass through a certain route. To do the copy of the index table in the system, in order to increase the system's fault-tolerance.

The rest of this paper is structured as follows. Section 2 discusses related work. Section 3 describes our design. Section 4 states the algorithm complexity and the experimental results. Section 5 discusses the improvements of our system. And finally, Section 6 concludes.

## 2. RELATED WORK

In the structured P2P networks, as every data in the DHT has a unique identifier, which is calculated by the consistent hash function. So keyword search must be based on the DHT.

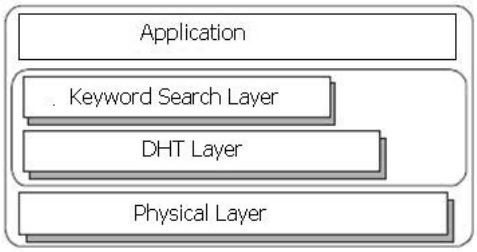


Figure 1: System Model

As it is described in the Figure 1，We divided P2P-based software system into four levels [6]. The physical layer is the bottom, DHT layer is above it. The keyword search services we provide is based on the DHT layer, no matter what DHT algorithm is.

Some approaches of distributed index table have been proposed, including Bloom filters [5] based on the Tapestry. The method, proposed in the [6], is to mapped keywords to a peak of Hypercube, which can be more efficient to find the mutli-keywords.

The concept of Trie tree in the [7] is as follows. A Trie or a prefix tree is an ordered tree. This data structure is used to store an associative array where the keys are usually strings. Unlike a binary search tree, no node in the tree stores the key associated with that node; instead, its position in the tree shows what key it is associated with. All the descendants of any one node have a common prefix of the string associated with that node, and the root is associated with the empty string. Values are normally not associated with every node, only with leaves and some inner nodes that happen to correspond to keys of interest.

## 3. Design

We modified the structure of the Trie tree, to a two-lay structure. We use the node which corresponds to the first letter as the main index, and use the node which corresponds to the sub-tree as the general index. The sequence of keywords which are directly stored in, need to have a common prefix. The DHTID of data are stored in the collection which corresponds to its keyword.

### 3.1. Structure of Tree

In our virtual network there are four types of nodes, namely the main index node, the general index node, the backup index node and the non-index node. We build a distributed Trie tree to index the keywords by the first two types of nodes. We map the alphabetical sequence into a linear space, for example from the A to Z. The main index node is in charge of the first level index, and will not index the real keywords, but only index the sub-tree which corresponds to its initial letter. The real keywords kept in the general index node, which take their common prefix as the INDEX_ID.

As the structure of the tree described in the Figure 2, there are six nodes as the main index node.They index the initial of the existent keywords separately. In the global routing table (we talked it in the Section 3.2) there are six main index nodes as the table items: INDEX_B, INDEX_C, INDEX_G, INDEX_O, INDEX_T, INDE_XW. The general index node INDEX_BL, INDEX_BU are kept in the main index node INDEX_B. The BL is the common prefix of the general index node INDEX_BL, and the node is likely to keep the keywords such as "block".

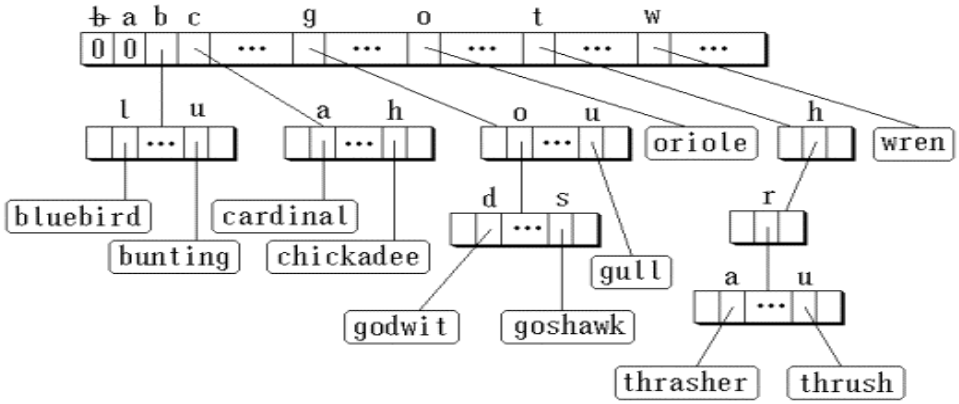


Figure 2: The Structure of Trie Tree

## 3.2. Routing Table

Table 2: Routing table item

| indexid | ip | port |
|---|---|---|
| INDEX_BL | 192.168.0.1 | 10000 |

We use a number nodes of high stability in the overlay as the main index table, the routing table that they consist is called global routing table. Every node in overlay whether is the index node or the non-index nodes must preserve the global routing table. And the main index node, put down the addresses of its sub-trees (the general index node), as the local routing table. The table item is depicted in the Table 2.

## 3.3. Search Scheme

Through its own global routing table, a node could find the corresponding main index node. Then according to the local routing table of the main index node, it will locate the keyword on which general index node. It will return the result after the completion of the looking for. If it is found that the keyword is inconsistent the routing table, the routing table must be synchronized. And then find again. There will be some redundant keywords or DHTIDs caused by the failure of node. If it finds that the DHTID corresponding to the keywords does not exist in the current overlay, it should remove the DHTID from the tree. As the same as the method of delete data.

For the multi-keywords search, it can carry out the above process for each keyword, and will end up with getting some different sets of DHTIDs. Their common ground is the final result.

## 3.4. Adjustment of Index Node

We add or delete index nodes according to the load. The addition of a general index node behaves like a “pull " process as follows. When the new node is joining in, it will send message to the original node which is heavy load for pulling index data. Secondly, split the sub-tree into two trees. Then one of the two trees includes the corresponding DHTID sets will be assigned to the new node, and removed from the original node. Finally notify the main index node to alter its local routing table. The process to delete a general index node is similar to the process of addition, but it's a "push" process initiated by the node which will be removed from the index tree. The main index node usually dose not changed unless the node fails.

It should be noted that the process of add dose not mean the node join in the DHT overlay, it is merely the process of a node from the non-index node turn into an index node, and vice versa.

## 3.5. Changes of Data

When the data in overlay is changing, the keyword of the data in the new node should also add into tree. To calculate the DHTID of the new data, then extract the keyword. If there is already the keyword in the tree, just insert the DHTID into the sets corresponding to its keywords. Otherwise create a new leaf node to join the corresponding sub-tree.

If the data is removed from the overlay, we should search the every keyword of the data, and then delete the DHTID in every set corresponding to the keywords. Moreover if there is not other DHTID in the set, we should also remove the keyword from the tree.

### 3.6. Fault -Tolerant Mechanisms

When the main index nodes find that the local general index node is failure, in order to ensure that the corresponding sub-tree on the failure node is not lost, it will open of the backup node immediately. When the general index nodes find their father (the main index node) is not active, the backup node will be opened of immediately. The newer broadcasts the global routing table to alter their global routing table, and then all the main index table will broadcast the local routing table to alter their global routing tables. Because the keywords and the DHTIDs of the lost data still exist in the global tree at this time, we should remove them when DHT search fails.

To ensure that there is no routing error in the search process, once a node finds its routing table error, it must launch to synchronize the routing table to the relative node (index node is best).

In order to find whether the node is failure as early as possible, the main index node and its local index nodes verify the validity of each other by sending heartbeat message. When the network change, the target of the message send to will be also change.

In our system, we make backup node for every index table, include the main and local index. Once the index node change, the backup node must synchronize to them.

## 4. Algorithm Analysis and Experimental Renaults

From the above we can see that the search process is mainly divided into two steps. First find the main index node by the global routing table, and then find the corresponding general index node according to the local routing table on the main index. Finally we will find the DHTID set corresponding to the keyword. It is easy to see that to locate the node which stores the keyword will take a total of only two jumps. If using the multi-level index mechanism (Section 5 will be mentioned), the process of locate a keyword need to jump several times. The overall number of jump is the depth of the distributed tree. There is an exception that we will search the keyword on the index node. In this case it will jump less times. The time required to search is the sum of the time required to jumping in the network and the local search time.

The local search time has already been mentioned in the [13], we will discuss the Average Search Length (ASL) of jump times as follows.

If there are *m* main index nodes, and *n* general index nodes in the Trie tree. And the number of keywords stored in the index node is *num* averagely. The total number of the node in the network is much greater then the sum of index nodes. The average search length without taking any circumstances to optimize into account is

$$\begin{aligned} \text{ASL} &= \frac{(n-1)\cdot num}{n\cdot num}\times 2 + \frac{num}{n\cdot num}\times 1 \\ &= \frac{(n-1)}{n}\times 2 + \frac{1}{n}\times 1 \end{aligned}$$

When there are a large number of keywords in the network, usually *n* is a large number. As the load of a node is limited, so the *num* is a constant.

$$\lim_{n\to\infty} ASL = 2$$

If using the multi-level index mechanism，there is $num_i$ keywords stored in the $i_{th}$ node. The depth of the tree in the $i_{th}$ node is $d_i$.

The total number of the keywords is:

$$sum = \sum_{i=1}^{n} num_i$$

Then the Average Search Length is:

$$ASL = \sum_{i=1}^{sum} \frac{1}{sum} \cdot \left( \frac{(n-1) \cdot num_i}{n \cdot num_i} \times d_i + \frac{num_i}{n \cdot num_i} \times 1 \right)$$

$$= \sum_{i=1}^{sum} \frac{1}{sum} \cdot \left( \frac{(n-1)}{n} \times d_i + \frac{1}{n} \times 1 \right)$$

So the average number of jump is:

$$\lim_{n \to \infty} ASL = \sum_{i=1}^{sum} \frac{1}{sum} \times d_i$$

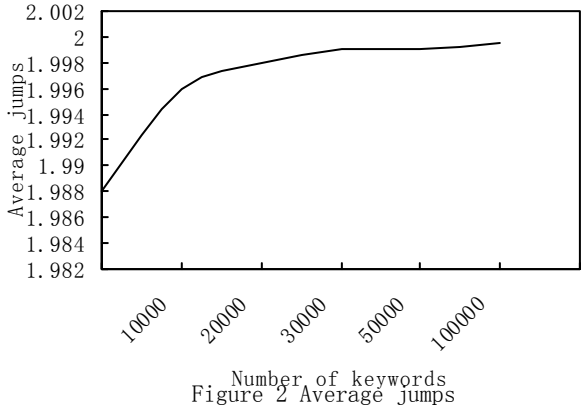


Figure 2 Average jumps

We simulated the process of searching one hundred times on a node, under such circumstances there are one hundred keywords on every node. As it is depicted in Figure 2, the average jumps trends to be *2*.

## 5. Improvements

One optimization method is to modify the sub-tree according to the search frequency of data and node, the number of keywords stored in the node and the shape of the tree at the time of adding the data or nodes. Increasing the number of the general index node to split the sub-tree in order to balance the load. In addition there is too little keywords or the lower frequencies of search in the node, merge its sub-tree with the neighbor's, and remove it from the distributed Trie tree. Afterwards it only use to as a non-index node.

We can also use another method as follows. Because most of the time the main index node is the only way in the process of search, we can start to use its backup node to share the load of main index node.

Using different divide methods to Trie tree can reduce the depth of the tree, so to the local routing table. For example we can use the initial character to devide to divide the tree at first. Then you can use the last character to divide the subtrees, and so on.

We can also use cache mechanism to accelerate the search speed. There is an index cache in the node, and you can cache the corresponding DHTIDs to search the high frequency keywords.

The backup index node can use as the secondary index node. It is also recorded on the routing table. In the process of search, the node could select the way according to the RTT (Round-trip Time).

We can use the multilevel index to reduce the space of the routing table and the keyword set. The main index node can point to the middle index node, and there is no keyword on it, so to the middle index node. It can constitute a multilevel index. However the routing table will become even more complicated, the jumps of search will be increased. Maintenance will be more difficult.

As the changes in the main index table cannot be notified to them at first time, it only will be detected when search fails. So the non-index node can update the routing table regularly to ensure the quick response of system.

## 6. Conclusion

In this paper we use in distributed Trie tree to store the keywords index. It needs only two jumps to locate the keyword. And it is a parallel process for multi-keywords search. So under such circumstances search is still high efficient. In addition it can be easy to insert or remove in the Tire tree, and only requires inserting or removing the corresponding branch of the tree. In order to load balance, we can also modify the structure of the tree as mentioned above.

However there will be redundant keywords and DHTIDs in the system. When an index node failure occurred, its index will not be lost. Because it would enable the backup node at this time, so the entire structure of the Trie tree does not change. As the node fails, the data in the node is lost. But their DHTIDs and keywords are still in the tree. And they become garbage. Our method will detect them after the error search in the next time. This is not the best method, because the garbage will exist for a long time in the system. However, the way to use updating index regularly would be to spend too much time.

## 7. 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,"in Proc. *ACM SIGCOMM'01*, Sept. 2001.

[2] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," *LNCS*, pp. 329–350, Nov. 2001.

[3] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in Proc. *ACM SIGCOMM'01*,pp. 161–172, Aug. 2001.

[4] 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 JSAC*, vol. 22, pp. 41–53, Jan. 2004.

[5] M. Mitzenmacher, "Compressed Bloom filters," *IEEE/ACM TON*,vol. 10, pp. 604–612, Oct. 2002.

[6] Yuh-Jzer joung,Li-Wei Yang,Chien-Tse Fang,"Keyword Search in DHT-based Peer-to-Peer Networks",*IEEE JOURNAL ON SELECTED AREARS IN COMMUNICATIONS*.VOL.25.NO.1,JANUARY 2007.

[7] http://en.wikipedia.org/wiki/Trie

[8] O. D. Gnawali, "A keyword-set search system for peer-to-peer networks," Master's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, United States, June 2002.

[9] F. Zhou, L. Zhuang, B. Y. Zhao, L. Huang, A. D. Joseph, andJ. Kubiatowics, "Approximate object location and spam filtering on peer-to-peer systems," in Proceedings of the 2003 ACM/IFIP/USENIX International Middleware Conference (Middleware 2003), ser. *Lecture Notes in Computer Science*, vol. 2672. Springer-Verlag, 2003, pp.1–20.

[10] P. Reynolds and A. Vahdat, "Efficient peer-to-peer keyword searching,"*in Proceedings of the 2003 ACM/IFIP/USENIX International Middleware Conference (Middleware 2003)*, ser. Lecture Notes in Computer Science, vol. 2672. Springer-Verlag, 2003, pp. 21–40.

[11] C. Tang and S. Dwarkadas, "Hybrid global-local indexing for efficient peer-to-peer information retrieval." in Proceedings of the *First Symposium on Networked Systems Design and Implementation (NSDI 2004)*.USENIX, 2004, pp. 211–224.

[12] S. Shi, G. Yang, D. Wang, J. Yu, S. Qu, and M. Chen, "Making peer-to-peer keyword searching feasible using multi-level partitioning," in Proceedings of the *3rd International Workshop on Peer-to-Peer Systems (IPTPS 2004)*., ser. Lecture Notes in Computer Science, vol. 3279. Springer-Verlag, 2005, pp. 151–161.

[13] Knuth D E.*The Art of Computer Programming*, volume 3/ Sorting and Searching. Addison-Wesley Publishing Company, Inc., 1973.