Introduction to the Kademlia protocol principle

MMX, K7mmx@tom.com

2006.2.20

Welcome to visit my baby park

Http://spaces.msn.com/members/cqmmx/

I. Introduction

The Kademlia Agreement (hereinafter referred to as Kad) is PetarP. Maymounkov and David Mazieres of New York University.
A study published in 2002 "Kademlia: A peerto -peer information system based on
The XOR metric.

Simply put, Kad is a distributed hash table (DHT) technology, but compared to other DHT implementation techniques, such as
Chord, CAN, Pastry, etc., Kad established a kind of unique metric based on XOR.
The new DHT topology greatly improves routing query speed compared to other algorithms.

After the famous BiTtorrent in May 2005 implemented the DHT technology based on the Kademlia protocol in version 4.1.0,
Soon, domestic BitComet and BitSpirit also implemented BitTorrent-compatible DHT technology.
Trackerless download method.

In addition, emule has also implemented a similar technology based on Kademlia very early (called DHT in BT, also called in emule)
Kad, note the difference from Kad in this article), and the difference between Kad technology used by BT software is key, value and
The node ID is calculated differently.

Second, the node status

In the Kad network, all nodes are treated as leaves of a binary tree, and the location of each node is determined by its ID.
The shortest prefix of the value is uniquely determined.

For any node, you can decompose the binary tree into a series of consecutive subtrees that do not contain your own. most
The high-level subtree consists of the other half of the tree that does not contain its own tree; the next subtree consists of the rest without its own
Half composition; and so on, until the whole tree is split. Figure 1 shows how node 0011 divides the subtree:

Figure 1: Subtree partitioning for node 0011

The dotted line contains the subtrees, with the prefixes from top to bottom being 0,01,000,0010.

The Kad protocol ensures that each node knows at least one node of its subtrees, as long as these subtrees are not empty. In this premise
Underneath, each node can find any node by ID value. The process of this routing is through the so-called XOR
Or) the distance is obtained.

Figure 2 demonstrates how node 0011 finds node 1110 through a continuous query. Node 0011 passes in step by step
The underlying subtree continuously learns and queries the best nodes, gets closer and closer nodes, and finally converges to the target node.

Figure 2: Targeting the target node by ID value

It should be noted that only the node 101 of the first query is already known by the node 0011, and the subsequent steps are queried.
The nodes are the nodes closer to the target returned by the previous query. This is a recursive operation.

Third, the distance between nodes

Each node in the Kad network has a 160-bit ID value as the identifier, and Key is also a 160-bit flag.
Each computer that joins the Kad network is assigned a node ID in the 160-bit key space.
Value (it can be considered that the ID is randomly generated), and the data of the <key, value> pair is stored in the ID value "most" close to the key valu
On the node.

Judging the distance between two nodes x, y is based on a mathematical XOR binary operation, $d(x, y) = x \oplus y$,
When the corresponding bit is the same, the result is 0, and the result is 1. E.g:

0 10 **1** 01
XOR **1** 10 **0** 01
----
1 00 **1** 00

Then the distance between the two nodes is 32+4=36.
Obviously, the difference in numerical values at the high level has a greater impact on the results.
For XOR operations, there are some mathematical properties as follows:
l $\mathbf{d(x, x) = 0}$

Page 4

l $\mathbf{d(x, y) > 0}$, **if** $\mathbf{x \neq y}$
l $\forall \mathbf{x, y : d(x, y) = d(y, x)}$
l $\mathbf{d(x, y) + d(y, z) \geqq d(x, z)}$
l $\mathbf{d(x, y) \oplus d(y, z) = d(x, z)}$
l $\forall \mathbf{a \geqq 0, b \geqq 0, a + b \geqq a \oplus b}$

Like Chord's measure of clockwise rotation, the XOR operation is unidirectional. For any given node x
And the distance $⊿ \geqq 0$, there will always be a precise node y, so that $d(x, y) = ⊿$. In addition, unidirectionality also ensures that
All queries with the same key value will gradually converge to the same path, regardless of the starting node position of the query.
In this way, as long as the <key, value> pair is cached along the nodes on the query path, the popular key value can be lightened.
The pressure of the node can also speed up the query response.

Fourth, K barrel

Kad's routing table is constructed from tables called K buckets. This is a bit like Tapestry technology, which

The routing table is also constructed in a similar way.

For each 0≦i≦160, each node is saved with some distance from itself and in the interval [twenty two) Some inside

Node information, which consists of a list of (IP address, UDP port, Node ID) data (Kad network is based on UDP protocol exchange information). Each such list is called a K bucket, and each K bucket internal information is stored. The positions are arranged in the order in which they were last seen, **recently (least-recently)** seen on the head, and **finally (most-recently)** seen in the tail. Each bucket has no more than k data items.

A list of all K buckets for a node is shown in Table 1:

I	distance	neighbor	
0	[0 1 twenty two)	(IP address, UDP port, Node ID)	0-1
		......	
		(IP address, UDP port, Node ID)	0-k
1	[1 2 twenty two)	(IP address, UDP port, Node ID)	1-1
		......	
		(IP address, UDP port, Node ID)	1-k
2	[2 3 twenty two)	(IP address, UDP port, Node ID)	2-1
		......	
		(IP address, UDP port, Node ID)	2-k
......			
i	[i i+1 twenty two)	(IP address, UDP port, Node ID)	I-1
		......	
		(IP address, UDP port, Node ID)	Ik
......			

160	[160 161 twenty two)	(IP address, UDP port, Node ID)	160-1
		......	
		(IP address, UDP port, Node ID)	160-k

Table 1: K barrel structure

However, usually when the value of i is small, the K bucket is usually empty (that is, there are not enough nodes, such as when i When =0, there may be at most 1 item); when the value of i is large, the number of items corresponding to K barrel is likely to exceed k (Of course, the wider the coverage distance, the greater the likelihood of having more nodes.) Here k is to balance system performance. A constant set with the network load, but must be an even number, such as k=20. In the implementation of BitTorrent, take The value is k=8.

Since the range of the coverage distance of each K bucket increases exponentially, this **forms more information about the nodes closer to itself. There is less information about nodes far away from you, which ensures that the route query process is converged.** Because it is divided by an exponential method In the meantime, it has been proved that for a Kad network with N nodes, only the logN step query is required at most. Determine the bit to the target node. This feature and the principle table of the distance table of the node on the Chord network. like.

When node x receives a PRC message, the IP address of sender y is used to update the corresponding K bucket. The steps are as follows:

1. Calculate the distance between yourself and the sender: d(x,y) = x ⊕ y, note: x and y are ID values, not IP addresses
2. The corresponding K bucket is selected by the distance d to perform the update operation.
3. If the IP address of y already exists in this K bucket, move the corresponding item to the end of the K bucket.
4. If the IP address of y is not recorded in the K bucket

(1) If the record of the K bucket is less than k, then directly put y (IP address, UDP port, Node ID) Information insertion queue tail

(2) If the record of the K bucket is greater than k, select the record entry of the header (if it is the node z) to perform RPC_PING operating

1 If z does not respond, remove the z information from the K bucket and insert the information of y into the tail of the queue.
2 If z responds, move the z information to the end of the queue and ignore the y information.

The K bucket update mechanism is very efficient to implement a strategy for updating recently seen nodes, unless the online node is Straight out of the K barrel. That is to say, nodes with long online time have a higher possibility to remain in the K bucket list.

This mechanism is based on the results of a study of a large number of user behavioral habits on the Gnutella network. The probability is inversely proportional to the online duration, as shown in Figure 3 (the abscissa is minutes and the ordinate is probability):

Page 6

Figure 3: Probability of online duration and continued online in the Gnutella network

It is obvious that the longer the user is online, the higher the likelihood that he will continue to be online in the next period.

So, by leaving nodes with long online time in the K bucket, Kad significantly increases the nodes in the K bucket at the next time. The probability that the segment is still online, which corresponds to the stability of the Kad network and reduces the network maintenance cost (no need Routing tables) bring great benefits.

Another benefit of this mechanism is that it can defend against DOS attacks to a certain extent, because only when the old node fails, Kad The K bucket information is updated, which avoids flooding routing information by joining new nodes.

In order to prevent the K bucket from aging, all K buckets that have no update operation within a certain period of time will be separately from their Randomly select some nodes to perform the RPC_PING operation.

These K-bucket mechanisms allow Kad to mitigate traffic bottlenecks (all nodes do not perform a large number of update operations at the same tim It also responds quickly to node failures.

Five, Kademlia protocol operation type

The Kademlia protocol includes four remote RPC operations: PING, STORE, FIND_NODE, FIND_VALUE.

1. The role of the **PING operation** is to probe a node to determine if it is still online.
2. The role of the **STORE operation** is to notify a node to store a <key, value> pair for later querying.
3. The **FIND_NODE operation** uses a **160-bit** ID as a parameter. The recipient of this operation returns what it knows more
Information about the IP address (UDP port, Node ID) of the K nodes close to the target ID.

The information for these nodes can be obtained from a single K bucket or from multiple K buckets (if closest to the target)
The K bucket of the ID is not full). In either case, the recipient will return information for the K nodes to the operator. But as
If the node information of all K buckets does not add up, then it will return information of all nodes to the initiator.

4, **FIND_VALUE operation** and FIND_NODE operation is similar, except that it only needs to return a node (IP
Address, UDP port, Node ID) information. If the recipient of this operation receives a STORE operation of the same key, then
Will return the stored value directly.

Note: In the Kad network, the data stored by the system is stored as a <key, value> pair. According to the author's analysis,

BitSpirit's DHT implementation, whose key value is the info_hash string of the torrent file, its value value and torrent
The documents are closely related.

To prevent forgery of addresses, the receiver is required to respond to a random 160-bit ID value in all RPC operations.
In addition, in order to be sure of the sender's network address, the PING operation can also be attached to the recipient's RPC reply message.

Sixth, routing query mechanism

One of the biggest features of Kad technology is the ability to provide a fast node lookup mechanism and also through parameters.
Find the adjustment of the speed.

If node x wants to find a node with an ID value of t, Kad performs a route lookup as follows:

1. Calculate the distance to t: $d(x,y) = x \bigoplus y$
2. Extract the information of the α nodes from the [log d] K buckets of x ("[""]" is the rounding symbol)
   FIND_NODE operation. If there is less than α information in this K bucket, choose the distance from the nearest multiple buckets.
   A total of α nodes close to d.
3. For each node that receives the query operation, if it finds that it is t, then it is the closest to t;
   Otherwise, measure the distance between yourself and t, and select the information of α nodes from its corresponding K bucket to x.
4. X performs the FIND_NODE operation again on each newly received node. This process is repeated until
   Each branch has a node response that is closest to t.
5. Through the above search operation, x obtains k node information closest to t.

Note: The term "closest" is used here because the node with the ID value of t does not necessarily exist in the network.
It is said that t is not assigned to any computer.

Here α is also a parameter set for system optimization, just like K. In the BitTorrent implementation, the value
Is α=3.

When α=1, the query process is similar to Chord's hop-by-hop query process, as shown in Figure 4.

Figure 4: Query process with α=1

The entire routing query process is recursive, and the process can be expressed as a mathematical formula:

$n_0 = x$ (the initiator of the query operation)

$N_1 = Find\_node_{n_0}(t)$

$N_2 = Find\_node_{n_1}(t)$

......

$N_l = Find\_node_{n_{l-1}}(t)$

This recursive process continues until $N_l = t$, or There is no information about t in the routing table of N_l, that is, the query failure

Since each query can get information from a K bucket that is closer to t, this mechanism guarantees that every recursive operation Ability to at least halve the distance (or reduce the distance by 1 bit) to ensure that the convergence speed of the entire query process is O(logN), where N is the number of all nodes in the network.

When node x wants to query the <key, value> pair, it is similar to the operation of finding the node. x selects k ID values closest to the key. The value of the node, perform the FIND_VALUE operation, and repeat the FIND_VALUE operation for each new node returned, straight

Return a value to a node.

Once the FIND_VALUE operation is successfully executed, the <key, value> pair will be cached in the most unreturned value. Close to the node. This way the next time you query the same key, you get the results faster. In this way, Popular <key, value> gradually expands the data cache, making the system very responsive, as shown in Figure 5.

Figure 5: Cache Principle

Seven, data storage

The process of storing <key, value> pairs of data is:

1. The initiator first locates k nodes whose ID value is closest to the key;
2. The initiator initiates a STORE operation on the k nodes.
3. The k nodes that perform the STORE operation republish all their <key, value> pairs of data every hour.
4. In order to limit the invalidation information, all <key, value> pairs of data expire after 24 hours of initial release.

In addition, in order to ensure the consistency of data distribution and search, it is stipulated at any time when node w finds new node u ratio Some <key,value> on w are closer to the data, then w copies these <key,value> pairs to u, but Will not be removed from w.

Eight, nodes join and leave

If node u wants to join the Kad network, it must be obtained with a node already on the Kad network, such as w. contact.

u First insert w into your appropriate K bucket, then perform a FIND_NODE operation on your own node ID, then Then update your K bucket content based on the received information.
Through the step-by-step query of the neighboring nodes from near and far, u finished
It becomes the construction of the K bucket information that is still empty, and also publishes its own information to the K bucket of other nodes.

In the Kad network, the routing table of each node is represented as a binary tree, and the leaf nodes are K buckets, which are stored in the K bucket Is the node information with the same ID prefix, and this prefix is the location of the K bucket in the binary tree. In this way, each K bucket All cover a part of the ID space, and all the information of the K bucket adds up to cover the entire 160-bit ID space, and There is no overlap.

Taking node u as an example, the routing table generation process is:

1. Initially, u's routing table is a single K bucket, covering the entire 160bitID space, as shown in Figure 6.
   Routing table
2. After learning the new node information, u will try to insert the information of the new node according to its prefix value into the corresponding In the K bucket:
   1 If the K bucket is not full, the new node is directly inserted into this K bucket;
   2 If the K bucket is full,

(1) If the K bucket coverage contains the ID of node u, split the K bucket into two identical sizes New K bucket and redistribute the node information in the original K bucket according to the new K bucket prefix value
(2) If the K bucket coverage does not have the ID of the packet node u, the new node information is directly discarded.

3. The above process is repeated and will eventually form a routing table for the structure of Table 1.
There are more information about the nodes that are close to each other
The result of less information from the far node ensures that the route query process can converge quickly.

Figure 6: Routing table generation evolution for node 000

In Figure 7, it demonstrates how the K bucket whose coverage contains its own ID value is gradually split.

Figure 7: K bucket splitting process for node 0100

When the K bucket 010 is full, the K bucket is split into two new ones because its coverage contains the ID of the node 0100.
K buckets: 0101 and 0100, the information of the original K bucket 010 will be redistributed into the two new K buckets according to their prefix values
Note that the 160-bit ID value notation is not used here, just for the convenience of the demonstration of the principle, in the actual Kad network.
The ID values are all 160bit.

Nodes leave the Kad network without releasing any information. One of the goals of the Kademlia protocol is to be able to work flexibly.
Any node that fails at any time. To do this, Kad requires each node to periodically publish all of its own storage.
<key, value> pairs the data, and caches the data in its own k nearest neighbors, so that it is stored in the failed node.
The data is quickly updated to other new nodes.