A Modification on the Chord Finger Table for Improving Search Efficiency

Lei Shi, Jing Zhou, Qi Huang, Wei Yan
School of Computer Science
Communication University of China
Beijing, China
{shilei_cs, zhoujing, hq_pinqy, yanweiyw}@cuc.edu.cn

Abstract—**Chord is a well-established and classical Peer-to-Peer (P2P) protocol for its simplicity and high search efficiency. There are many studies about how to further improve Chord search efficiency. In this paper, we first discuss the finger table to analyze the reason why Chord has high search efficiency. Based on the analysis, we made a modification on the item *start* in the finger table of Chord to improve search efficiency. Results from theoretical analysis and experiments show that the modification improves search efficiency as we anticipated.**

Keywords—Chord; search efficiency; improvement; the finger table; start

I. Introduction

Over the last few years, there has been a large volume of research on Peer-to-Peer (P2P) network. The P2P network is an alternative to the client/server system for sharing resources, e.g. files. A P2P network has a robust, distributed and fault tolerant architecture. Basically, there are basic two types of P2P networks: structured P2P networks and unstructured P2P networks. Each of them has its own applications and advantages.

One of the most elegant advantages of structured P2P network is that search is deterministic: given the key of an object, the search algorithm guarantees to find the object. Among structured P2P networks, Chord [1] has been researched and applied widely because of its many advantages such as simplicity, high efficiency and credibility. Chord is simple, routing a key through a sequence of $O(\log N)$ other nodes toward the destination. A Chord node requires information about $O(\log N)$ other nodes, which are organized into a table called the finger table, for efficient routing, but performance degrades gracefully when that information is out of date. This is important in practice because nodes will join and leave arbitrarily, and consistency of even $O(\log N)$ state maybe hard to maintain. Only one piece information per node need be correct in order for Chord to guarantee correct (though slow) routing of queries; Chord has a simple algorithm for maintaining its information in a dynamic environment.

However, there are many issues with Chord [2], one of which is how to reduce the search path length so as to improve search efficiency. This paper focuses on how to further improve Chord search efficiency. Since the finger table is one of the key elements in making Chord highly efficient, in this paper, a simple modification on item *start* of the finger table, is proposed to further improve the search efficiency.

This work is funded by the Engineering Disciplines Planning Project of the Communication University of China (No.XNG1246, 132013XNG1320), and the National Natural Science Foundation of China (No. 61103198).

The remainder of the paper is organized as follows. Section 2 briefly introduces Chord, and especially analyzes why the finger table can make Chord search highly efficient. Section 3 describes our modification on the finger table, and theoretically analyzes why our modification can improve the search efficiency. Section 4 presents the experimental results. Section 5 provides a survey of related work. Finally, we conclude and outline items for future work in Section 6.

II. Chord

Since our paper proposes a modification on the Chord finger table to further improve search efficiency, prior to detailing our work, it is necessary to briefly introduce Chord, and especially analyze why the finger table can make Chord search highly efficient.

Chord assigns each node and key an m-bit ID using a base hash function such as SHA-1. IDs are ordered in an ID circle modulo 2^m. Key k is assigned to the first node whose ID is equal to or follows the ID of k in the ID space. This node is called the *successor node* of key k, denoted by *successor*(k). If IDs are represented as a circle of number from 0 to 2^m-1, then *successor*(k) is the first node clockwise from k around the ID circle, as Fig. 1 shows.

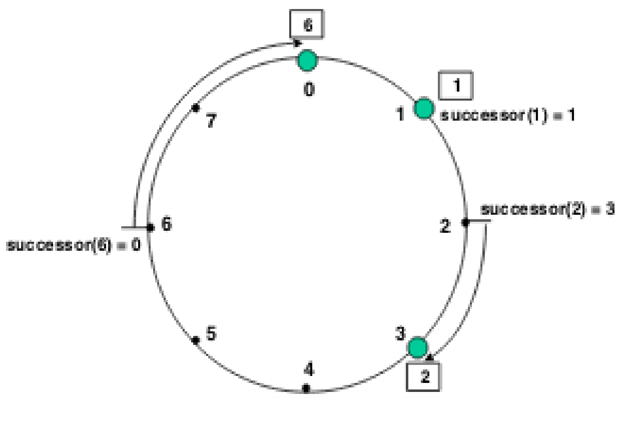

Fig. 1. An ID circle consisting of three nodes 0, 1,and 3, where key 1 is assigned to node 1, key 2 to node3, and key 6 to node 0.

In order to accelerate the search for *successor*(k), each node, n, maintains a routing table with m entries, called the *finger table*. The ith entry in the finger table at node n contains the ID and IP address of the first node, s, that succeeds n by at least 2^{i-1} on the ID circle, i.e., $s = successor((n + 2^{i-1}) \bmod 2^m)$, $1 \le i \le m$. Chord denotes s by *finger*[i].*node*, and $(n + 2^{i-1}) \bmod 2^m$ by *finger*[i].*start*, i.e.,

$$finger[i].start = (n + 2^{i-1}) \bmod 2^m.$$

For example, $finger[1].start = n + 2^{1-1} = n + 2^0 = n + 1$,

$finger[2].start = n + 2^{2-1} = n + 2^1 = n + 2$,

$finger[3].start = n + 2^{3-1} = n + 2^2 = n + 4$,

$finger[4].start = n + 2^{4-1} = n + 2^3 = n + 8$,

...

Suppose the range of the ID space is [0, N]. When node n wishes to resolve a query for *successor*(k), n first searches its finger table for the closest predecessor p of k, and forwards its query to p. Suppose that k is in $finger[i].interval = [finger[i].start, finger[i+1].start)$. Since $(finger[i].start - n) / (finger[i + 1].start - n) = 2^{i-1}/2^i = 1/2$, this means that the distance from p, i.e., *finger*[i].*node* to k is at most half the distance from n to k. Thus, by repeating forward operation, the distance between the node handling the query and k halves in each step. Finally, after at most log N hops (or forwardings), the distance between the node handling the query and k will be one, that is, we have arrived at *predecessor*(k).

III. Modification

How can we make the search for *successor*(k) more efficient? A simple idea is that, if in the finger table, $(finger[i].start - n)/(finger[i + 1].start - n) > 1/2$, the distance from *finger*[i].*node* to k will decrease to at most *less than* half the distance from n to k, and the search efficiency is expected to be improved to some extent. Thus, we propose a modification on the item *start* in the finger table as below.

$$finger[i].start = [n + 2^{i-1} + (i-1)^2] \bmod 2^m.$$

First, we calculate the values of modified *finger*[i].*start*, (i = 1, 2, 3, ...) as follows.

$finger[1].start = n + 1 + 0^2 = n + 1$,

$finger[2].start = n + 2 + 1^2 = n + 3$,

$finger[3].start = n + 4 + 2^2 = n + 8$,

$finger[4].start = n + 8 + 3^2 = n + 17$,

$finger[5].start = n + 16 + 4^2 = n + 32$,

$finger[6].start = n + 32 + 5^2 = n + 57$,

...,

$finger[12].start = n + 2048 + 11^2 = n + 2169$,

$finger[13].start = n + 4096 + 12^2 = n + 4240$,

...

Next, we calculate the values of (modified *finger*[i].*start* – n) / (modified *finger*[i + 1].*start* – n), (i = 1, 2, 3, ...) as follows.

$(finger[1].start - n) / (finger[2].start - n) = 1/3 \approx 0.3333$,

$(finger[2].start - n) / (finger[3].start - n) = 3/8 \approx 0.3750$,

$(finger[3].start - n) / (finger[4].start - n) = 8/17 \approx 0.4705$,

$(finger[4].start - n) / (finger[5].start - n) = 17/32 \approx 0.5312$,

$(finger[5].start - n) / (finger[6].start - n) = 32/57 \approx 0.5614$,

...,

$(finger[12].start - n) / (finger[13].start - n) = 2169/4240 \approx 0.5115$,

...

Finally, we compare $(finger[i].start - n) / (finger[i + 1].start - n)$ original and modified (i = 1, 2, 3, ...) as Fig. 2 shows.

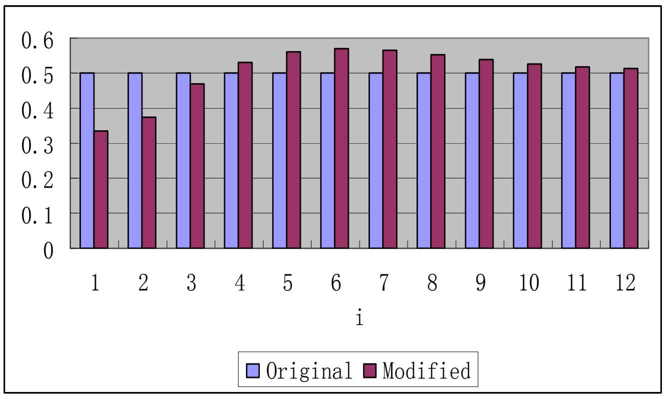

Fig. 2. $(finger[i].start - n) / (finger[i + 1].start - n)$ compared with original and modified.

It can be observed that,

1. When $i \le 3$, (modified *finger*[i].*start* – n) / (modified *finger*[i + 1].*start* – n) < 1/2;

2. When $i > 3$, (modified *finger*[i].*start* – n) / (modified *finger*[i + 1].*start* – n) > 1/2. As should be pointed, although (modified *finger*[i].*start* – n) / (modified *finger*[i + 1].*start* – n) can be observed closer and closer to 1/2 with the increase of i, whenever $i > 6$, the ratio > 1/2 nevertheless. For example, when $i = 24$, the ratio is 0.500014 > 1/2.

Suppose that the ID length is m bits. This means that when node n wishes to resolve a query for *successor*(k) and search the closest predecessor p of k in its finger table,

1. If k lies in (n, modified *finger*[4].*start*) = (n, $n + 17$), the distance from p (i.e., *finger*[*i*].*node*) to k will decrease to *more than* half the distance from n, which makes search inefficient than the original.

2. If k lies in [modified *finger*[4].*start*, $2^m - 1$] = [$n + 17$, $n + 2^m - 1$],. the distance from p (i.e., *finger*[*i*].*node*) to k will decrease to at most *less than* half the distance from n, which makes search more efficient than the original.

Obviously, $i = 4$ is the turning point of search efficiency after modification. Suppose that the ID length is m bits, in the finger table of node n, the length ratio of (n, modified *finger*[4].*start*) / (modified *finger*[4].*start*, $(n + 2^m - 1) \bmod 2^m$), where $(n + 2^m - 1) \bmod 2^m$ is the farthest k clockwise from n around the Chord circle, is $[(n + 17) - n] / [(n + 2^m - 1) - (n + 17)] = 17 / (2^m - 18)$. Now we calculate the length ratio with the increase of m as follows.

$m = 6$, $17/(2^6 - 18) = 17/(64 - 18) = 17/46 \approx 0.369$,

...,

$m = 10$, $17/(2^{10} - 18) = 17/(1024 - 18) = 17/006 \approx 0.016$,

...,

$m = 13$, $17/(2^{13} - 18) = 17/(8192 - 18) = 17/8174 \approx 0.002$,

...

It can be observed from figures above that, with the increase of m, the length ratio of (n, modified *finger*[4].*start*) / (modified *finger*[4].*start*, $(n + 2^m - 1) \bmod 2^m$) will be less and less, closer and closer to 0. This implies that, if the ID length m is large enough, for quite many searches for *successor*(k), k is very possible to lie in [modified *finger*[4].*start*, $(n + 2^m - 1) \bmod 2^m$] of the node handling the search. Thus, if a node performs a number of searches for *successor*(k), where k is randomly picked, the mean number of hops per search is expected to be less than the original.

Note that the number of entries of the finger table after modification is still m, the same as the original, and the maintenance cost such as node join (including initializing finger table of newly joined node and updating finger tables of existing nodes) is about the same as the original.

IV. Experiments and Analysis

In order to verify the efficiency after the modification, we simulated networks with different ID space length. We varied the ID length from 3 to 13 and conducted a separate experiment. In each experiment, we picked a set of keys randomly to search successors, and measured the Mean Hop Number (MHN) using the original and modified *start*, as Table 1 and Fig. 3 show.

TABLE I. A COMPARISON OF THE MEAN HOP NUMBER BETWEEN THE ORIGINAL AND MODIFIED START

m	3	4	5
Original	1.506	2.001	2.502
Modified	1.742	2.235	2.642
Modified – Original	0.236	0.234	0.140

m	6	7	8
Original	2.994	3.523	4.005
Modified	2.938	3.319	3.735
Modified – Original	– 0.055	– 0.203	– 0.270

m	9	10	11
Original	4.503	5.000	5.513
Modified	4.184	4.641	5.126
Modified – Original	– 0.319	– 0.359	– 0.387

m	12	13
Original	6.018	6.519
Modified	5.594	6.082
Modified – Original	– 0.424	– 0.437

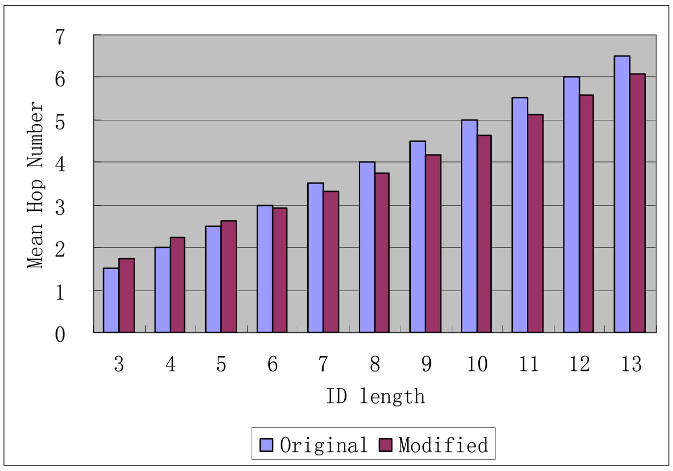

Fig. 3. Mean Hop Number compared with *start* original and modified.

It can be observed that, when the ID length ≤ 6, MHN original < MHN modified. However, with the increase of the ID length, when the ID length > 6, MHN original > MHN modified. This experimental result shows that, when the ID length > 6, search using the modified *start* is certainly more efficient than the original, which verifies our anticipation in previous section.

In a practical network, the ID length must be large enough to make the possibility of two nodes or two keys hashing to the same ID negligible, such as 120-bit ID by SHA-1, which is much larger than 6. Therefore, our modification is feasible in practical network.

V. Related Work

Up to now, there are many improved schemes to improve search efficiency of the Chord protocol.

In order to be able to more fleetly forward search message to the target node, Wen [3] proposed a Chord algorithm of order k, but the length of the routing table almost increases linearly with the increase of k.

Gupta A et al. [4, 5] proposed a One-Hop algorithm, by which the step of search is only one. Though search efficiency of this algorithm is very high, the cost of each node to maintain is very large.

Jiang et al [6] replaced every original unidirectional link with a bidirectional link and added a converse finger table to improve search performance of Chord. However, with the churn in the P2P system, the algorithm can't effectively enhance the search performance.

In response to these problems, Dai et al. [7] established a Chord network using information of neighbor's neighbor, and proposed a Chord search algorithm NN-Chord based on links of neighbor's neighbor. On the basis of no more networks spending, their algorithm can find an object node which is closer to key. However, the search area only covers 3/4 Chord circle every time, meanwhile, this algorithm only can process unidirectional lookup.

Fan et al. [8] proposed a search algorithm BNN-Chord (Bidirectional NN-Chord) based on NN-Chord, adding the coverage and finger density of routing table and repetitious entries are deleted.

VI. Conclusion and Future Work

Chord is a well-known P2P protocol for its simplicity and high efficiency. In order to further improve search efficiency of Chord, in this paper, we propose a modification on the item *start* in the finger table of Chord, using $finger[i].start = [n + 2^{i-1} + (i-1)^2] \bmod 2^m$, rather than $(n + 2^{i-1}) \bmod 2^m$. Results from theoretical analysis and experiment show the modification improve search efficiency.

Although our modification can further improve search efficiency, there are still some issues that need to be addressed in the future.

1. As Figure 2 shows, with the increase of i, (modified $finger[i].start - n$) / (modified $finger[i+1].start - n$) is closer and closer to 1/2. Thus, when i is fairly large, the ratio will be almost equal to that of original finger table. Therefore, the search efficiency when i is fairly large, should be studied.

2. In this paper, we propose a modification on *start* of the finger table. Nonetheless, are there any other modifications on the finger table, or even new designs on the finger table, which can improve search efficiency better? Such issues should be studied as well.

References

[1] I. Stoica, R. Morris, D. Karger, F. Kaashoek and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup service for internet applications," in Proceedings of ACM SIGCOMM '01, San Diego, CA, Aug.2001: 161-172.

[2] Sylvia Ratnasamy, Scott Shenker and Ion Stoica, Routing Algorithms for DHTs: Some Open Questions.

[3] Wen Zhongzhi. Chord of High Order: AN New P2P Lookup Policy. Computer applications technology of Sichuan university.2005.

[4] Gupta A, Liskov B, Rodrigues R. One hop lookup for peer-to-peer overlays[C]//Proceedings of the 9th Workshop on Hot Topics in Operating Systems(HotOS-IX), 2003.

[5] Gupta A, Liskov B, Rodrigues R. Efficient routing for Peer-to-Peer overlays[C], in Proceedings of 1st Symposium on Networked Systems Design and Implementation (NSDI'04), San Rancisco, California,2004-03

[6] Jun-Jie Jiang, Fei-Long Tang, Feng Pan, Wei-Nong Wang. Using bidirectional links to improve peer –to-peer lookup performance. Journal of Zhejiang University SCIENCE A, 2006,7(6):945-951.

[7] DAI Bin, Wang Furong, Jianhua Ma and Liu Jian. Enhanced Chord-Based Routing Protocol Using Neighbors' Neighbors Links. IEEE, 2008.pp. 385-390.

[8] Fan chao, Hongqi Zhang, Xuehui Du, Chuanfu Zhang,. Improvement of Structured P2P Routing Algorithm Based on NN-Chord. in Proceedings of Wireless Communications, Networking and Mobile Computing (WiCOM), 2011 7th International Conference:1-5.