

Improving Routing Load Fairness in Structured P2P Overlay Networks

You ZHOU, Keiichi KOYANAGI

Graduate School of Information, Production and Systems, Waseda University, Japan

zhouyou@ruri.waseda.jp, keiichi.koyanagi@waseda.jp

Abstract— Structured P2P overlay networks provide rather balanced query routing load compared to centralized network systems. Despite their distributed and scalable design, issues such as different in-degrees of peers, peer churn and non-uniform request distribution may lead to poor routing load fairness in the overlay. In this paper, we propose an enhanced routing strategy that dynamically selects next-hop destination based on peers' current load information and the characteristics of the routing load distribution in the overlay network. Our approach can fairly balance the routing load among close neighbors as well as diverting a portion of the routing load from heavily loaded areas to less loaded ones. Simulation results show that our proposal significantly improves the routing load fairness among peers while the query performance remains almost the same.

Keywords— overlay network, routing algorithm, load balancing, Chord

I. INTRODUCTION

Structured P2P overlay networks provide efficient data storage and retrieval mechanisms in a distributed and scalable manner. Generally speaking, the load is distributed among all peers and thus it is more balanced than systems using centralized architecture.

However, certain inherent designing issues and external factors might lead to very unbalanced routing load distribution in overlay networks. Firstly, peers in the same overlay network have considerably different in-degrees (i.e. number of incoming routing table entries), which cause them to experience imbalanced routing load. This difference is caused by the impact of peer churn and the randomness of interval lengths between peers in the key address space. Another major issue is non-uniform request distribution. Common structured overlay networks are designed to have a balanced load under the assumption that the request distribution is uniform, in other words, all data objects are equally popular. While this assumption seems reasonable for certain scenarios, in most real world P2P systems such as file sharing P2P applications, data objects often have very different popularity. Observations show that the actual request distribution in P2P applications roughly follows the Zipf's Law [4]. Consequently, peers near the most popular data objects will bear much more routing load.

The issue of balancing the routing load in structured P2P overlay networks is extensively studied in the literature. A common approach is to improve routing load balance by

minimizing the difference of peers' responsible intervals. In [5], M. Bienkowski et al. employ methods to manipulate the peer ID generating procedure to ensure peers' interval lengths differ at most by a constant factor. Thus the inherent cause of load imbalance is mitigated. Another kind of solutions uses the concept of virtual servers [6]. In [7] proposed by P. B. Godfrey and I. Stoica, each physical node in the network instantiates one or more virtual servers with different IDs to act as peers in the overlay. When a node becomes heavily loaded, it transfers some of its virtual servers to a proper node with fewer loads. However, the above proposals incur high maintenance overhead and extra complexity. More importantly, they are still based on the assumption of a uniform request distribution.

There are studies that put emphasis on dealing with highly skewed request distribution. In [8], G. Yuan et al. assumed a Zipf-like request distribution and proposed a modified version of Pastry [2] that dynamically remove heavily loaded neighbors from routing table and replace it with less loaded peers sharing the same ID prefix. By removing popular peers from others' routing table, their routing load will be reduced. However, it also increases query hops and the failure rate due to the removal. R. Cuevas et al. proposed an enhanced Chord protocol called e-chord [9] that selects finger entry from a consecutive range of peers with random probability in order to have balanced in-degrees between peers. Even under non-uniform request distribution, this approach still has acceptable routing load fairness.

In this paper we address the issue of routing load imbalance in structured P2P overlay systems such as Chord [1], Pastry [2] and Kademlia [3], etc. We analyze the factors that lead to imbalance and propose corresponding strategies to improve the routing load fairness. Our proposal assumes a highly skewed request distribution. Zipf's Law is used to model the request distribution in the overlay network and Jain's Fairness Index [10] is used as the evaluation metric for routing load fairness. We use Chord as the example to implement our proposal and then run simulations extensively to verify the effectiveness of our proposal.

The rest of the paper is organized as follows. In Section II, we analyze inherent and external causes of poor routing load fairness in structured overlay networks. Section III includes an introduction to Jain's Fairness Index and explains how we quantify the routing load fairness in simulations. Section IV

describes the proposed strategies to enhance Chord. Section V is the simulations results and evaluation. Finally, in Section VI we conclude our study and point out possible directions for future work.

II. ISSUES IN STRUCTURED OVERLAY NETWORKS

In this section we analyze two important factors that cause routing load imbalance, one is an inherent design issue in structured overlay networks, and the latter is the external cause for load imbalance.

A. Different In-degrees of Peers

In common structured P2P overlay networks, the IDs of participating peers are derived from hash functions which theoretically produce a uniformly random distribution. But as the result of randomness, the intervals between peers in the key address space may vary with considerable difference. Analysis from [1] indicates that the longest interval could be $O(\log N)$ times longer than the shortest interval (N denotes the number of peers in the overlay). By the deterministic algorithm with which the peer selects its routing table entries, the probability of a certain peer being chosen as a routing table entry by distant peers is proportional to the length of its corresponding interval. Thus ultimately, peers will have rather different number of incoming routing table entries (i.e. in-degree).

What is more, the dynamic property of P2P network environment, or churn, meaning the continuous arrival and departure of peers, will further aggravate the difference of in-degrees: peers joining and leaving the overlay will change the interval length of their closest neighbors and impact their in-degrees. Also, newly joined peers need time before they can be known to distant peers and be used as routing table entry, so new peers are likely to have much smaller in-degrees than others.

From the basic principles of network, we know that this imbalance of in-degrees will result in a serious imbalance of query routing load because a peer with a bigger in-degree is likely to receive much more routing queries from others. Even if some peers are very close in key address space, their routing load will vary considerably. This is quite counter-intuitive and unbalanced.

B. Non-uniform Request Distribution

In the logarithmic querying mechanisms of structured overlay networks such as Chord, querying process is carried out by recursively forwarding the query message to a peer whose ID is closer to the target key in the key address space. Thus queries are forwarded into progressively smaller areas of the whole overlay network during every hop. From this we can infer that for a certain data object, the routing load for this object is bigger when the peer is closer to the target key.

The imbalance distribution of routing load to one certain data object can be cancelled out when we assume a large number of objects with similar popularity, i.e. each object is requested at more or less the same frequency. This is commonly assumed in numerous studies. However, the

request distribution is in fact highly skewed in real world P2P systems such as file sharing systems. In [4], observations show that the actual request distribution in many P2P systems follows the *Zipf's Law*.

Zipf's Law is an empirical law of statistics that can be briefly described as follows: when the most popular object is requested C times, the i^{th} popular object is requested approximately C/i times. For example if the most popular file in a file sharing system is requested 1,000 times, then the 2nd popular object is requested $1000 / 2 = 500$ times, the 3rd popular object is requested around $1000 / 3 = 333$ times, and so on.

With a highly skewed non-uniform request distribution and the imbalance characteristic of querying, peers near the most popular objects will be heavily loaded.

III. ROUTING LOAD FAIRNESS METRIC

We would like to clarify that in this paper the notion of routing load of a peer is represented by the number of queries it received and subsequently forwarded to other peers. This is counted by each peer itself.

For the purpose of quantifying the routing load fairness, we introduce a widely accepted metric called *Jain's Fairness Index* [10]. It is used to evaluate the fairness of a distribution and is calculated using the following formula:

$$JFI = \frac{(\sum_{i=1}^n x_i)^2}{n \sum_{i=1}^n x_i^2}$$

We apply the calculation of JFI on the number of routed queries to represent the extent of routing load fairness. As the formula implies, the value of JFI can vary from $1/n$ to 1, where 1 indicates the optimal fairness (the number of routed queries are the same for every peer) and $1/n$ means the worst case (the entire routing load is placed on one single peer).

IV. ROUTING LOAD FAIRNESS METRIC

For the purpose of improving the overall routing load fairness in the overlay network, we proposed two strategies that respectively aim at balancing the routing load among close neighbors and redirecting part of the routing load from peers near popular objects to peers far from them. We implemented those strategies on Chord [1] because it is widely used and considered as one of the most representative structured overlay network protocols.

The following is a brief description of the Chord protocol's mechanisms: Chord has a logical ring topology. IDs of both peers and data objects are generated using hash functions and have the same bit length (later referred to as m). Data objects are stored in the first peer succeeding their ID. Chord protocol designed a unique routing table called finger table which has m entries. The i^{th} entry stores the first peer that succeeds the $\text{FingerID}_i = (\text{PeerID} + 2^{i-1}) \bmod 2^m$. In other words, the longest finger spans approximately 1/2 of the Chord ring, the second longest spans 1/4 of the ring, etc. (Fig. 1)

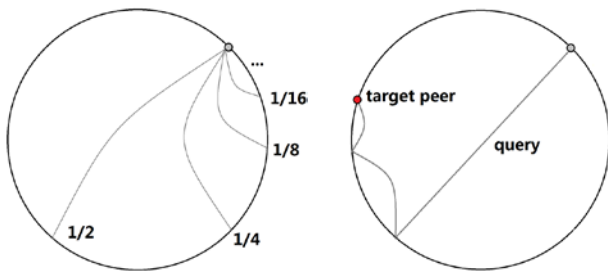


Figure 1. Chord finger table (left) and query example (right)

When a peer searches for a data object, it specifies the object's ID and then select the finger peer whose ID is closest to the target ID as the next-hop destination of this query hop. The query is recursively forwarded and ultimately reach the peer hosting the object with the target ID. (Fig. 1)

A. Balancing Routing Load Among Close Peers

As mentioned in Section II, one major cause of poor routing load fairness is the different in-degrees, even if peers are very close in the overlay, their routing loads can be very unbalanced.

The following is the strategy that tries to evenly distribute the routing load among close neighbors by extending the definition of finger table and modifying the deterministic next-hop selection algorithm in Chord.

1) Enhanced finger table

Except the information already stored in a standard Chord finger table (i.e. peer ID and IP addresses of each finger peer), our enhanced finger table also stores the quantified routing load information and a list of finger peer's immediate successors. An example of the contents in the enhanced finger table is shown in Fig. 2

Finger	Peer ID	IP Addr.	Load	SuccList
1	4D00A43B	*.*.*.*	103	
2	4D17DA7E	*.*.*.*	142	
3	4D5A7AEE	*.*.*.*	287	
...	...	...	...	...

Successor List of Finger[2]		
Peer ID	IP Addr.	Load
4D19201B	*.*.*.*	126
4D23E78E	*.*.*.*	166
4D24B90A	*.*.*.*	143
...	...	...

Figure 2. Enhanced finger table

These new information fields are updated along with standard finger table maintenance. When peers send periodic maintenance messages (e.g. keep-alive), the information of the current load and successors is included in the communication. By appending extra data onto maintenance messages we only have to increase the communication cost slightly, meanwhile the freshness of the information about load and successors is guaranteed.

2) Next-hop selection algorithm

In the original Chord, when a peer tries to select a next-hop peer to forward the query, it deterministically chooses the finger peer who is closest to the target ID. Instead of selecting that particular finger peer, in our strategy we compare the peer

and its successors using the new information stored in the enhanced finger table, and then select the least loaded one from them as the next-hop destination. The pseudo-code is shown below (Algorithm 1):

Algorithm 1: Enhanced next-hop selection

```

find_next_hop (targetID) {
    next_hop = null;
    for all fingers from long to short {
        if (finger[i].ID == targetID)
            return finger[i];
        if (finger[i].ID ∈ (currentPeer.ID, targetID)) {
            next_hop = finger[i];
            for each successor in finger[i].succ {
                if (finger[i].succ[j].ID == targetID)
                    return finger[i].succ[j];
                if (finger[i].succ[j].load < next_hop.load)
                    next_hop = finger[i].succ[j];
            }
            break;
        }
    }
    return next_hop;
}

```

With these enhancements on the finger table and routing algorithm, although peers may not directly be chosen as distant peers' finger table entries, they can still be selected as next-hop destination and receive forwarded queries. Through this, the in-degrees of nearby peers are indirectly averaged. And since the sender always select the least loaded one from peers in that small area to forward the query, the routing load of those peers are balanced.

Another merit is that the average number of query hops will be slightly reduced because now the peer also holds the information on fingers' successors, and it is possible to directly find out the target peer without one more hop through the finger.

B. Diverting Routing Load to Distant Areas

Logarithmic hop distance and the greedy selection of the closest finger are the key factors that provide fast and scalable key-value search mechanism in Chord, but they inevitably incur the imbalance problem mentioned in Section II. We intend to modify the greedy selection algorithm for next-hop finger

The basic principle of our strategy is that after we find the finger nearest to the target key (e.g. the k^{th} one in finger table), we also consider the $(k-1)^{\text{th}}$ finger, which is approximately half the finger length. We select the least loaded peer among k^{th} and $(k-1)^{\text{th}}$ finger and all of their successors as the next-hop destination.

As a result, peer will not always greedily select the longest possible finger but occasionally take a shorter finger. As is shown in Fig. 3, the upper blue arrows represent original greedy finger selection path, and the lower white arrows represent a possible alternative finger path in our strategy. In the lower case the query is forwarded to a peer that is far away

from the target key instead of a peer close to it (the upper case). From a broader perspective, this kind of alternative paths will divert a portion of the routing load out of the area near the target key, which generally has more loads.

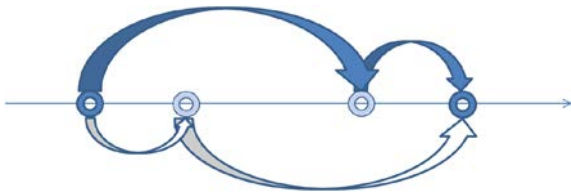


Figure 3. Alternative finger paths

Since the strategy takes shorter fingers, it will, to some extent, increase the number of hops a query needs. There is a trade-off between routing load fairness and query performance. And because we know that if the selected finger is already short, taking even shorter finger will not result in much difference. This strategy will only be reasonably effective when selecting long fingers. So a good choice in the trade-off is to only apply this strategy to K longest fingers, for example $K = 3$.

V. SIMULATION AND EVALUATION

We implemented our proposal based on Chord protocol and conducted simulations using PeerSim [11], which is a flexible and lightweight P2P network simulator. We carried out a series of simulations in order to verify the effectiveness of the two strategies respectively. For simplicity, the strategy described in Section IV-A is referred to as Range Finger Table (RFT) and the other described in Section IV-B is referred to as Shorter Finger (SF).

A. Network Parameters and Request Distribution

In the following simulations, a virtual network of 10,000 peers ($N=10,000$) is created. Peer ID length is set to 32 bits and successor list is 16.

We assume a non-uniform request distribution that follows the Zipf's Law. Specifically speaking, the top 500 popular objects are considered ($N_{\text{Object}}=500$) and the most popular data object is queried a million times ($Q_1=1,000,000$). Queries are initiated at random peers in the overlay network and the number of queries for each popular object is calculated according to the Zipf's Law

The K parameter mentioned in SF is set to 3, so only when the original algorithm intends to use the 3 longest fingers, the enhanced algorithm will consider a shorter finger and their successors.

B. Simulation Results

1) Routing load fairness of the most popular object

First we examine the routing load distribution of the most popular object in one simulation run. For clarity, only 100 peers preceding the peer hosting the most popular object is shown.

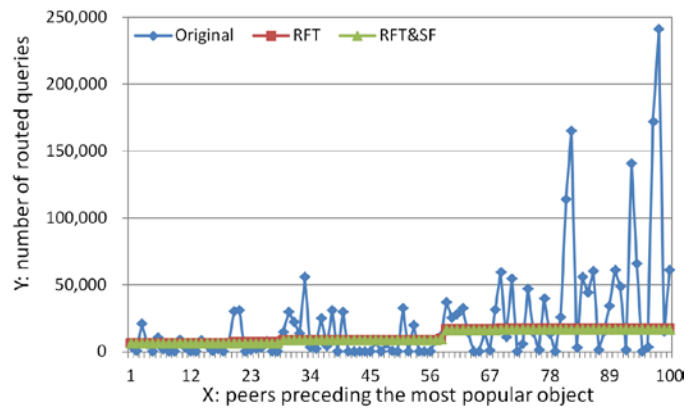


Figure 4. Routing load of 100 peers preceding the most popular object

In Fig. 4, the routing load of 100 preceding peers in the original Chord changes drastically without obvious pattern. The routing load varies in a wide range from 0 to as much as 250,000. A few peers are severely loaded comparing to others. By contrast, in our proposal the load is much more balanced and almost at the same level, the maximum routing load is less than 25,000.

Notice that due to the large scale of Y axis (number of routed queries) and the similarity of results, RFT and RFT&SF almost overlapped in this figure.

TABLE 1. PERFORMANCE (MOST POPULAR OBJECT)

Algorithm	Query hops	Local JFI
Original	6.644	0.248
RFT	5.458	0.854
RFT&SF	6.273	0.854

The average hops and the local Jain's Fairness Index of these 100 peers is shown in TABLE 1. We can see that the average query hops are not very different, but our proposals dramatically increased the JFI, meaning the load is much more balanced among peers. As for the comparison between RFT and RFT&SF, because this is the routing load from a very small area in the overlay network and SF is intended to move load in a larger scope, applying SF does not notably improve the fairness in this case.

2) Overall routing load fairness

Fig. 5 shows the routing load distribution in the entire overlay network. Routing loads are sorted by increasing order and are normalized by dividing the average routing load in each simulation run.

From the results we know that the routing load in the original Chord is very unbalanced. The minimum load is only 15% of the average while the maximum is about 50 times bigger than the average. On the other hand, in RFT and RFT&SF the loads are mostly very close to the average value (denoted as 1 in the Y axis), and the maximum load is just approximately 5 times the average. Also, from the figure we can see that RFT&SF yield better results than using RFT alone.

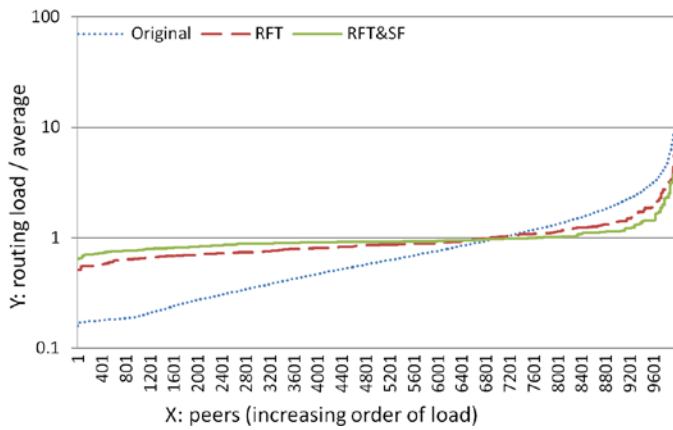


Figure 5. Routing load distribution in the entire overlay

TABLE 2. PERFORMANCE (THE ENTIRE OVERLAY)

Algorithm	Query hops	JFI
Original	6.557	0.263
RFT	5.457	0.759
RFT&SF	6.268	0.858

We performed 10 simulation runs for each strategy and calculated the average performance statistics as shown in TABLE 2. Our proposal only has small impact on query performance (actually reduced the average hops slightly) but has much higher JFI (increased from 0.263 to 0.858), this clearly indicates that the load is very balanced and routing load fairness has been improved very much.

VI. CONCLUSIONS

In this paper, we analyzed both inherent and external issues that lead to poor routing load fairness in structured overlay network. Then based on Chord we proposed an extended routing table definition and two enhancements on routing strategy in order to improve the routing load fairness under non-uniform request distribution. The effectiveness of our proposal is verified by analyzing routing load distribution and Jain's Fairness Index in simulations. The results shows that after applying our proposal, the routing load distribution is far more balanced than the original Chord, meanwhile the incurred maintenance cost is minimal and the average query hop remain $O(\log N)$.

Several aspects of this study can be improved with further efforts. Since the P2P network environment is largely heterogeneous, we might also consider fairly distributing the routing load proportionally according to the capacity of peers. Alternative ways of representing of routing load, such as using the rate rather than the total number of routed queries, may also be helpful in certain scenarios. These will be the directions of our future studies.

REFERENCES

- [1] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, et al., "Chord: a scalable peer-to-peer lookup protocol for Internet applications," *Networking, IEEE/ACM Transactions on*, vol. 11, pp. 17-32, 2003.
- [2] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," presented at the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, 2001.
- [3] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," presented at the First International Workshop on Peer-to-Peer Systems, 2002.
- [4] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," *SIGOPS Oper. Syst. Rev.*, vol. 37, pp. 314-329, 2003.
- [5] M. Bienkowski, M. Korzeniowski, F. M. a. d. Heide, and F. M. Heide, "Dynamic Load Balancing in Distributed Hash Tables," in *Proc. IPTPS*, 2005, pp. 217-225.
- [6] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with CFS," *SIGOPS Oper. Syst. Rev.*, vol. 35, pp. 202-215, 2001.
- [7] P. B. Godfrey and I. Stoica, "Heterogeneity and load balance in distributed hash tables," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, 2005, pp. 596-606 vol. 1.
- [8] G. Yuan, C. Ling, and T. Ke-Ming, "A Load Balancing Method under Zipf-Like Requests Distribution in DHT-Based P2P Network Systems," in *Proc. Web Information Systems and Mining, 2009. WISM 2009. International Conference on*, 2009, pp. 656-660.
- [9] R. Cuevas, M. Uruena, and A. Banchs, "Routing Fairness in Chord: Analysis and Enhancement," in *Proc. INFOCOM 2009, IEEE*, 2009, pp. 1449-1457.
- [10] R. Jain, W. Hawe and D. Chiu. "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems". DEC-TR-301, 1984.
- [11] A. Montresor and M. Jelasity, "PeerSim: A scalable P2P simulator," in *Proc. Peer-to-Peer Computing, 2009. P2P '09. IEEE Ninth International Conference on*, 2009, pp. 99-100.