An Adaptive Proximity Route Selection Scheme in DHT-Based Peer to Peer Systems*

Jiyoung Song, Sungyong Park, and Jihoon Yang

Dept. of Computer Science, Sogang University,
Seoul, South Korea
{itsuki, parksy}@sogang.ac.kr, jhyang@ccs.sogang.ac.kr

Abstract. This paper presents a proximity route selection scheme which is adaptive to the underlying network conditions. The proposed scheme repeatedly estimates and updates the total lookup latency based on the information from the neighborhoods. The updated information is used to select the next peer dynamically to route the query, which results in reducing the overall lookup latency. We implemented the scheme using the Chord, one of the most popular DHT-based peer to peer systems, and compared the performance with those of original Chord and the CFS' server selection scheme. The proposed scheme shows performance improvement over other schemes.

1 Introduction

Chord [1] is one of the most popular distributed hash table (DHT) based peer to peer systems which provides scalable services for the storage and retrieval of (key, data) pairs. In Chord, a hash function such as SHA-1 assigns a key (or identifier, ID) from the circular m-bit identifier space to each node and data. Each node maintains a routing table (finger table in Chord) of other live members (neighborhoods). The size of the routing table is either constant or $O(log\ p)$, where p is the number of participating nodes. The ith entry in the routing table of node n points to a node with the smallest identifier clockwise from $n+2^{i-1}$. During the query routing process, Chord selects the next routing peer in the finger table whose identifier is the closest to the data's key to minimize the total routing hops.

However, the overlay hops along the routing path may contain heavy links and the lookup latency increases although we have minimal routing hops. In order to solve this problem, CFS [3] proposes a server selection scheme for Chord on the domain of proximity route selection (PRS) [2]. In CFS, each node predicts the entire lookup latency which is calculated by using the total number of nodes and average overlay hop delay. Among these neighborhoods, one with the minimum cost is selected as the next routing peer. The problem of this approach is that it is hard to approximate the total number of nodes and the average hop latency from the local, since the number of neighborhoods for each node is too small. The inaccurate estimation may also result in decreasing the performance.

* This work was supported by grant No. R01-2003-000-10627-0 from the Basic Research Program of the Korea Science & Engineering Foundation.

K.-M. Liew et al. (Eds.): PDCAT 2004, LNCS 3320, pp. 778–781, 2004

In this paper, we propose an adaptive proximity route selection (APRS) scheme in order to reduce the total lookup latency by modifying the Chord protocol. Unlike CFS, our scheme estimates the query delivery time using previous estimation history, where the new estimated value is calculated by combining the observed latency and the old estimated value. This scheme does not require any knowledge of total number of nodes and the average hop latency.

The rest of the paper is organized as follows. Section 2 presents the adaptive proximity route selection scheme in detail. In Section 3, we evaluate the APRS scheme and present the experimental results. Section 4 concludes the paper.

2 Adaptive Proximity Route Selection Scheme (APRS)

The APRS scheme uses recursive lookup. Each node on the routing path forwards a query for a specific key to one of its neighborhoods and updates the estimation about the expected lookup latency based on the information from their neighborhoods unless the node is the predecessor whose ID is just previous to the key on the ID space. In what follows, we provide the detailed description of our scheme.

For a given node with ID n, if a key q of query in $[n+2^i, n+2^{i+1})$, where i is an integer between 0 and m-1 (on the m- bit identifier space), is received, we assume that node n is in i-state. (Similar to the term Range used in [5])

For the query q, the candidates set of the next routing peer in node n is determined as follows:

$$candidate\_set_n(q) = \{\text{Node } x \mid \forall x \text{ in the routing table which satisfies its ID} \in (n, q]\} \quad (1)$$

Note that the route is restricted by the candidate set.

Each node obtains the routing policy by predicting which one of $candiate\_set_n(q)$ can reduce the total lookup latency for routing the query. To do this, APRS requires additional information for each entry in the finger table. For each entry with ID x, the form is,

$$(x, \text{cost}_i, \text{cost}_{i+1}, ..., \text{cost}_m) \text{ for all entry in the routing table} \quad (2)$$

where the integer i starts at the state of neighborhood peer ID. The value of cost_i which is denoted by $Q_n(x,i)$ means the estimated lookup latency by selecting x of candidates for i-state query at node n. This extended table is called a lookup table.

The node periodically updates this lookup table by the rule shown below.

$$\begin{aligned} Q_n^{new}(x,i) &= (1-\alpha)\cdot Q_n^{old}(x,i) \\ &+ \alpha \cdot (\text{observed latency to } x + \min_{z \in \text{neighborhoods of x}} Q_x(z,i)) \end{aligned} \quad (3)$$

where α [†] is a constant between 0 and 1. This update rule (3) is drawn from Q-routing algorithm [4] based on reinforcement learning on the domain of distributed packet routing algorithms.

There are two strategies to update the lookup table by receiving the information[‡] from neighborhoods. According to the original Chord protocol, each node in the

[†] We set alpha to 0.5 in our simulation for experimental reason.

[‡] This is the second part given in equation (3).

system runs the stabilization protocol periodically in the background for neighborhoods to refresh its entries. It is possible to add additional information corresponding to the term $Q_x(z,i)$ and measure the latency to x based on the latencies observed while performing the background process. The advantage of our scheme is that no additional control message exchange is needed. However, due to this delayed updating, the adaptability to the network can be decreased. The other alternative is to update the lookup table by requesting the response message for the forwarded query, immediately. In our simulation, the latter is adopted for the fast convergence.

The decision rule of selecting the next peer to route the query is greedy. If node n is in i-state for the query q, the greedy decision rule is as follows:

$$\pi_n(i_q) = \underset{c \in candidate_n(q)}{\arg\min} \; Q_n(c, i_q) \tag{4}$$

where $\pi_n(i_q)$ is the routing policy for node n during the routing process.

3 Experimental Results

We have conducted our experiment over modified SFS-simulator [6] to implement our scheme. Two different topologies, ring and transit-stub random graph with Waxman model by GT-ITM [7], are used for underlying physical connectivity between nodes and we have compared the performance of our scheme (APRS) with those of unmodified Chord and the CFS' server selection scheme.

The simulation consists of three phases. First, each node joins one-by-one. At this time, each node selects its underlying network position in the topology and ID randomly. In the second phase, each node inserts about 50 documents (in average) into the system. In the third phase, we measure the lookup latency by generating 50 requests for randomly selected documents. During the simulation, we do not consider the dynamics of nodes (e.g. node joins and leaves) in the system explicitly.

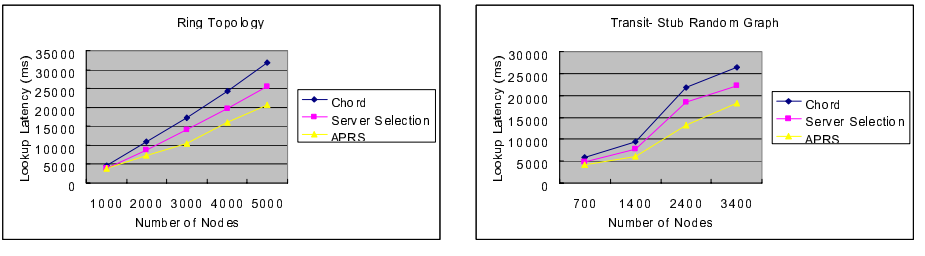

Fig. 1. Average lookup latency for Ring and Transit-Stub Random Graph

Fig. 1 shows the average lookup latency for the ring and transit-stub random graph by varying the number of nodes ranging from 1000 to 5000 and from 700 to 3400, respectively. APRS scheme shows performance improvement over the original Chord

and server selection method by about 31% and 20%, respectively. Note that server selection has the advantage of knowing the number of nodes in the system.

Fig. 2 shows the time required for the latency to converge into a certain level. The 1 time unit in Fig. 3 (i.e., x-axis) means the time when one request on all nodes in the system has been resolved. As we can see from Fig. 2, after about 21 requests are reached, the average lookup latency is converged to a certain level. This means that the proposed scheme adapts to the network changes within a reasonably short period.

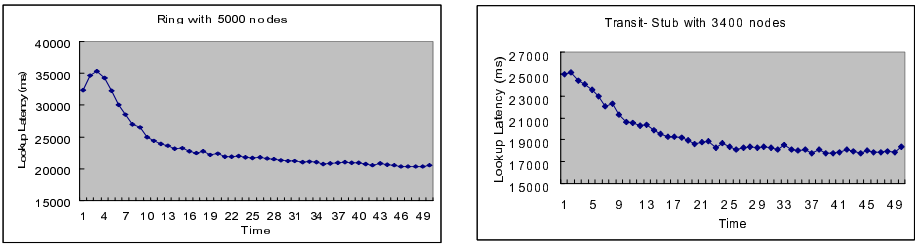

Fig. 2. Convergence time

4 Conclusion

In this paper, we have proposed an adaptive proximity route selection scheme. Nodes in Chord are able to obtain the distributed routing policy by using APRS. To show that, we have implemented and compared our scheme with the original Chord that does not aware of the network proximity and CFS' server selection through simulation. From the experiments, we showed that the performance of our proposed scheme was better about 20% and 31% than the server selection and original Chord, respectively without much network overhead and computation. We also showed that the period of adapting to network changes is also reasonably short.

References

1. Stoica, I. et. al. : Chord : A Scalable Peer-to-Peer Lookup Protocol for Internet Applications, IEEE/ACM Transactions on Networking Vol. 11 Issue 1 (2003) 17-32.
2. Gummadi, K. et. al. : The impact of DHT Routing Geometry on Resilience and Proximity, Proc. of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (2003) 381-394.
3. Dabek, F. et. al. : Wide-area cooperative storage with CFS, Proc. of the 18th ACM Symposium on Operating Systems Principles (2001) 202-215.
4. Boyan, J., Littman, M.: Packet routing in dynamically changing networks: A reinforcement learning approach, Proc. of Neural Information Processing Systems (1994) 982-988.
5. Zhang, H., Goel, A., Gobindan, R.: Incrementally Improving Lookup Latency in Distributed Hash Table, Proc. of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (2003) 114-125.
6. Chord Simulator, http://pdos.lcs.mit.edu/cgi-bin/cvsweb.cgi/sfsnet/simulator.
7. GT-ITM, http://www.isi.edu/nsnam/ns/ns-topogen.html.