빠른 노드 검색과 부하감소를 위한 새로운 가쉽기반 P2P 네트워크 알고리즘

축효위*, 장경식**

*한국기술교육대학교 전기전자공학과

**한국기술교육대학교 정보기술공학부

A Novel Algorithm for Fast Node-search and Redundancy Reduction in Gossip-based P2P Network

Xiao-Wei Zhu*, Kyung-Sik Jang**

*Dept. of Electrical and Electronic Engineering, Korea University of Technology and Education

**Dept. of Information Technology Engineering, Korea University of Technology and Education

E-mail : felixzoo@kut.ac.kr

요 약

가쉽 기반 프로토콜을 이용한 P2P 네트워크 급속히 발전하고 있다. 특히 그룹통신에서 는 가쉽 기반 프로토콜이 높은 신뢰성을 보장하고 확장성이 있다. 본 논문에서 제시하는 자기 조정 프로토콜은 사용 그룹 크기를 모르는 경우에 이웃에서 목록을 얻는다. 그리고 이웃 목록의 노트 백업 메커니즘을 이용해서 시스템의 부가적인 부하를 감소시킨다. 제안된 시스템 모델, 기반의 알고리즘과 시뮬레이션 평가 결과들을 본 논문에서 제시한다.

ABSTRACT

P2P networks are undergoing rapid progress and inspiring numerous developments by gossip-based protocol. Gossip-based protocols for group communication have attractive scalability and reliability properties. We propose a self-organizing algorithm in the sense that the size of neighbor list achieved without any node knowing the group size. We also propose an efficient mechanism to reduce the redundancy of the system by backing up the nodes in the neighbor list. We present the design, theoretical analysis, and a detailed evaluation of the proposed algorithm and its refinements.

키워드

P2P, gossip-based self-organizing neighbor list

Ⅰ. Introduction

With the development and expansion of internet-wide application, the construction of network has great changes as the need of reliable protocol is widely deployed from the traditional sever-client model to peer-to-peer model which is developed by gossip-based protocol. Peer-to-peer systems have many interesting technical aspects like decentralized control, self-organization, adaptation and scalability.

In this paper, we mainly focus on the pure p2p system which means each node in a p2p network has similar functionalities and plays the role of a server and a client at the same time. This provides immense flexibility for users to perform application-level routing, data posting, and information sharing on the Internet. We propose a scalable membership

protocol. This protocol can reduce the redundancy of network system and make a fast-searching of the nodes throughout the network but is achieved without any node knowing the group size. The protocol is simple, fully decentralized, and self-configuring. As the number of participating nodes changes, we show both analytically and through simulation that the size of partial views automatically adapts to the desired value.

As shown is figure 1, every node in this pure p2p system has a list called *neighbor list* which owns some nodes links as the size of the neighbor list. The messages will be gossiped among the nodes and their neighbors. For example, when a new node will join the network, it will firstly send a join message to the bootstrap node, and then the bootstrap node will assign the new node some nodes in its neighbor list. So the new node can gossip its joins message over the network and get the entrance with its own neighbor list.

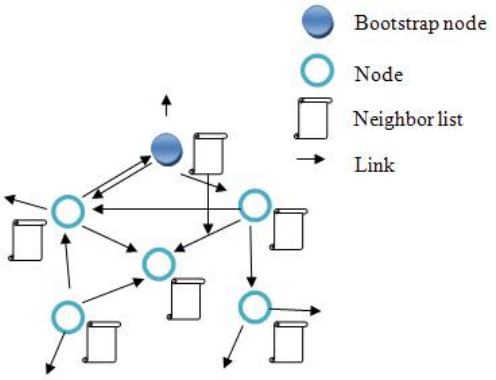

Fig.1. Network diagram

The remainder of the paper is organized as follows: We describe the membership protocol in Section2, including a sketch of the theory behind it. and we conclude in Section 3.

II. Design and optimization

Figure 2 depicts the join algorithm diagram of our protocol. As we can see the procedure of node jon, we will have a detailed description about that in the following part.

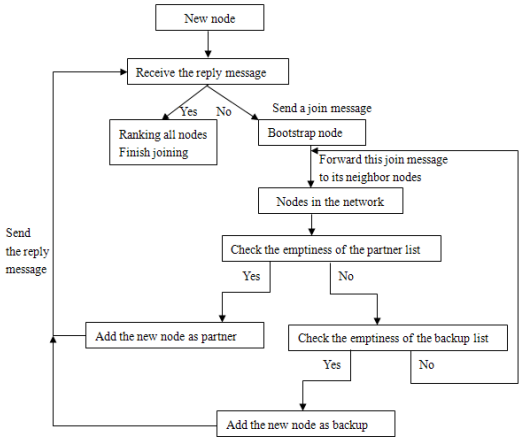

Fig.2. Depiction of the join algorithm diagram

A. Node joining and partner management

Each node has a unique identifier, such as its IP address or node ID, and maintains a partner list containing a partial list of the identifiers who have the best connection with this node, meanwhile, each node also maintains a backup list containing nodes backup for the nodes in partner list. An example of partner list and backup list is shown in figure 3. The criteria of making the lists depend on the wire speed and network condition. According to this performance, we will make the top N nodes into partner list, and the following M nods into the backup list.

In a basic node joining algorithm, a newly joined node first contacts the bootstrap node. According to the new node's IP address, response time (delay), wire speed and such network condition, the bootstrap will send a join message to its partners. Each message is a 4-tuple <*seqnum,id,timetolive,timestamp*>, where seqnum is a sequence number of the message, id is the node's identifier, time to live records the remaining valid time of message, time stamp is the real time when the message have any actions.

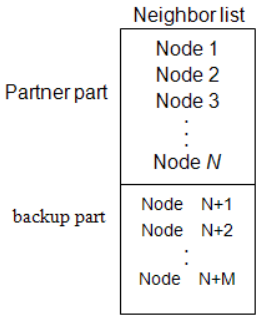

Fig.3. Neighbor list

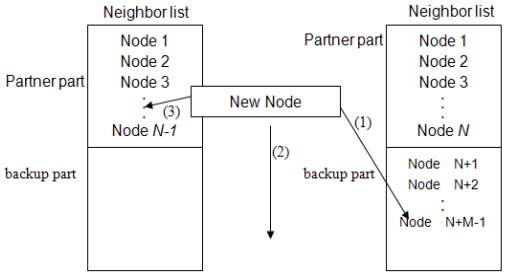

Fig.4. Node join

Since the nodes in backup list backup for the nodes in partner list, we have 3 cases when the node receives a join message: (1)partner list is full while backup list not; (2) partner list and backup list are both full.(3)partner list is not full and backup list is empty. As show in figure 4, for case (1), the new node will be added into the backup part after comparing its link condition with the others; for case (2), the joining message will be sent to nodes in the partner part; for case (3), node will be added directly to the backup part.

Once, a node in the network receiving this join message will check its partner list and backup list. The first thing the node will do is to compare the new node with the nodes in both lists. Second, if the new node's performance is better comparing with the node 1~N+M. the new node will be add into the list. If not; for case 1, the new node will be only added into the backup list. After being added, the new nodes will receive a reply message which has the same format as joining message.

If node does not accept the new node, it will forward the joining message to its partners and so on.

When TTL=0, this procedure will be over and, of course, the new node will receive several reply messages. According to the reply messages, the new node will also rank the performance and add the top 1 to N nodes into its partner list and the following M nods into the backup list。

s is the new node.
m is the size of neighbor list.
n is the size of partner list.
$T_d$isthenode'sdelaytime.

Algorithm1: Node join management
//forward a join message of node s to all the nodes p of the neighbor list
for all nodes p∈neighbor list **do**
send(s,join message) to p;
end for

Algorithm2: Handling a forwarded join message
//nodes p receiving s join message
if **s**∈neighbor list **then**
algorithm described in Algorithm 1;
else if p≠m **then**
neighbor list= neighbor list + {s}
else
for(j=0,j<m,j++)**do**
compare T_{ds}withT_{dpj};
if T_{ds}<T_{dpj}
neighbor list = neighbor list+{s};
delete p---$_{m;}$
else algorithm described in Algorithm 1;
endif
end if

B. Node departure and system recovery

In order to keep to stable size of node's partner list, we need all the nodes to keep contact with their partners and backup nodes always. Because our network actually is an asymmetric network, we should make each node to send a probe message in time T_P to detect whether its partner and backup nodes is in or out.

Now we are going to introduce the function of *backup list*.

Before this we have to declaim that the communication between nodes is based on gossip protocol, which means it will occur a lot of redundancy during each communication, in order to deduce the redundancy, we issue this backup list.

In the network, when node A sends a probe message and fine that a node in its backup list is gone, (there are still some nodes in backup list) it will do nothing, but if there is a node gone in the partner list, node A will make the first node in its backup list be its partner node, which means the backup node changes into the partner node. When all the nodes in backup list are gone, node A will initiate the recovery message to the bootstrap node similar to the join message to find the new proper nodes for partner and backup.

Due to the backup list, we don't have to gossip whenever there's a node in partner list is gone which will help to reduce the redundancy.

Algorithm3: system recovery
//node sending a probe to detect its neighbors
if p<n **then**
for all nodes p∈neighbor list **do**
send(join message) to bootstrap;
end for

C. Data searching and transmission

The backup list will not perform during the data searching and transmission. There are nodes in the partner list play the role of neighbors. When a node send a searching message for a certain data, it will issue a message of 5-tuple<seq num, id, time to live, delay time, contentseq>, where the first four items are same as the joining message, contentseq is the sequence of the data which will generated to a number by system. A node who wants a data will send this kind of message to its partner list. And the node who receives the message will check its own content list and forward this message to other nodes or original node by adding a similar message after the original message. As show in figure 5:

Original node	Seq0	nodeIP	TTL	Time0	ContentSeq
Transmit nodeA	Seq1	nodeIP	TTL-1	Time1	
Transmit nodeB	Seq2	nodeIP	TTL-2	Time2	
⋮		⋮			

Fig.5. message gossiping

If nodes received this message again, it will stop forwarding.

After doing this, the original node can receive the reply from the other nodes in the network and according to the message table, it can calculate the transmit time which means it can choose the best transmit path and find the inferior path as backup path.

Algorithm4: Data searching
//sending a searching message for data x
for all nodes p∈partner part **do**
send(searching message) to p;
end for

Algorithm5: handling a searching message
//receiving a searching message with TTL=t
if t>0;**do**
checking files;
if no file match the request **then**
forward (searching message = searching message + node information) ;
t-- ;
else send reply message = searching message + node information + content);
end if
else if t=0; **do**
checking files;
if no file match the request **then**
delete message;
else send reply message = searching message + node information + content);
end if
end if

Ⅲ. Conclusion

In this paper we propose the design and theoretical analysis of our protocol. The protocol provides each member a neighbor list which consists of partner and backup lists. With the help of this, node can have a fast searching of the nodes in the network and the network redundancy will be reduced due to the decrease of gossip time.

Reference

[1] Xinyan Zhang, Jiangchuan Liu, Bo Li, and Tak-Shing Peter Yum, "CoolStreaming/DONet: A Data-Driven Overlay Network for Efficient Live Media Streaming,"

[2] Ayalvadi J. Ganesh, Anne-Marie Kermarrec, and Laurent Massoulie"Peer-to-Peer Membership Management for Gossip-Based Protocols"IEEE TRANSACTIONS ON COMPUTERS, vol. 52, No. 2, February 2003

[3] Soontaree Tanaraksiritavorn and Shivakant Mishra"Evaluation of Gossip to Build Scalable and Reliable Multicast Protocol"

[4] Djamal-Eddine Meddour, Mubasher Mushtaq"Open Issuse on P2P Multimedia Streaming"