WUJNS
Wuhan University Journal of Natural Sciences

Vol.12 No.5 2007 887-892

Article ID: 1007-1202(2007)05-0887-06
DOI 10.1007/s11859-007-0006-z

Resource Search in Unstructured Peer-to-Peer System Based on Multiple-Tree Overlay Structure

□ **YU Jianqiao, LIAO Jianwei**

College of Computer and Information Science, Southwest University, Chongqing 400715, China

Abstract: We propose a multiple-tree overlay structure for resource discovery in unstructured P2P systems. Peers that have similar interests or hold similar type of resources will be grouped into a tree-like cluster. We exploit the heterogeneity of peers in each cluster by connecting peers with more capacities closer to the root of the tree. The capacity of a peer can be defined in different ways (e.g. higher network bandwidth, larger disk space, more data items of a certain type etc.) according to different needs of users or applications.

Key words: unstructured P2P system; tree-like cluster; similar interest

CLC number: TP 393.4

Received date: 2007-01-10

Foundation item: Supported by the National High Technology Research and Development Program of China (2006AA10Z1E6)

Biography: YU Jianqiao(1957-), male, Professor, research direction: database, information retrieval, artificial intelligence. E-mail:yujq@swu.edu.cn

0 Introduction

Unstructured P2P systems don’t maintain a logical interconnection structure among the peers. Lacking a logical structure, the most sensible way to search for data items in the system is to flood the requests to the peer nodes. There are two strategies to improve the search efficiency of an unstructured P2P system. The first strategy focuses on how queries are delivered in the P2P network[1]. The 2nd strategy is to connect the nodes in a certain way so that the desired resources may be found quickly, such as Gnutella[2].

This paper proposes a multiple-tree overlay structure for resource discovery in unstructured P2P systems. Peers that have similar interests or hold similar type of resources will be grouped into a tree-like cluster[3,4]. We exploit the heterogeneity of peers in each cluster by connecting peers with more capacities closer to the root of the tree. The capacity of a peer can be defined in different ways (e.g. higher network bandwidth, larger disk space, more data items of a certain type etc.) according to different needs of users or applications.

There are several research works on reducing the traffic overhead and improving search efficiency in Gnutella-like P2P networks[5-7]. We especially emphasize on constructing an efficient overlay as the underlying network for flooding.

1) Trail-based technique

FloodTrail[1] defines Trail of a flooding as a collection of P2P links, along which a query reaches a peer for the first time during a flooding. Links that are used to transmit redundant messages of that query are excluded from Trail.

However, in a more dynamic environment, peers constantly join in or depart from the P2P network. The Trail may not reflect current network topology. To keep the freshness of the Trail, a lease mechanism is used. A trail is invalid after the lease expires, thereafter the query will be flooded again to construct a new trail.

2) Tree-like sub-overlay structure

The design of LightFlood[8] is motivated by the observation those in pure flooding most redundant messages are generated when messages are flooded further away from the requester, while the coverage improves the most in the first few hops.

Therefore, LightFlood floods the request at low hops. During high hops, LightFlood will select a set of links to form a tree-like sub-overlay called FloodNet that organizes peers into a small number of low diameter clusters and lets messages be flooded in the sub-overlay.

3) Policy-based link structure

GUESS[9] investigates a new type of search architecture, in which messages are not forwarded, and peers have complete control over who receives its queries and when under the GUESS protocol, peers directly probe each other with their own query messages, rather than relying on other peers to forward the message.

In GUESS, there is a tradeoff between the query response time and the traffic overhead. Users must carefully set the policy of maintaining the link and query caches and choosing the time interval of selecting a new neighbor to send out the query.

1 System Design and Implementation

1.1 Overview

Our system is built as an infrastructure for message passing in an unstructured P2P network (e.g. Gnutella). Gnutella is a “search” protocol rather than a scheme for managing or maintaining the interconnection among peers in a P2P network. In our system, we regulate a set of rules for building connections over the peers. We define how a new peer joins the system, which peer this new peer should take as its neighbor, how the system is kept stable in face of highly dynamic failure in the P2P network.

The basic idea is that we sort each peer by the type of its resource and group the same type of peers to form a tree. Note that a peer can join different trees due to its different types of resources. When a peer issues a query, it will send the message to the root of the specific tree which maintains the requested type of resources. The root will deliver the query to all its children in a top-down fashion until TTL = 0 or the leaves of the tree are reached. Peers will respond a QUERYHIT to their parent if there are matched data items. Apparently, the root in each tree acts as an entry point of queries and bears most workload. Therefore, we move stronger peers to closer to the root. Compared with previous research to improve Gnutella, our system has several features listed below[10].

1) Traffic reduction: Based on our tree-like structure, queries are sent from the root of the tree and then passed to its children, which in turn forward queries to their children until TTL = 0. There is no redundant message delivered in the tree structure.

2) Exploiting the heterogeneity of peers: Our system will move nodes with higher capability closer to the root of the tree. Therefore, we can make use of more powerful peers to handle more works by locating them closer to the root.

3) Practical and trust: In pure flooding, a query is passed within a scope based on its TTL value. In our system, queries will be forwarded only to these peers holding the requested type of resources.

4) Fully-distributed and self-healing: we distribute the jobs to all peers participating in the system. Each peer only needs to spend some disk storage and network bandwidth for maintaining its connection with its parent and children.

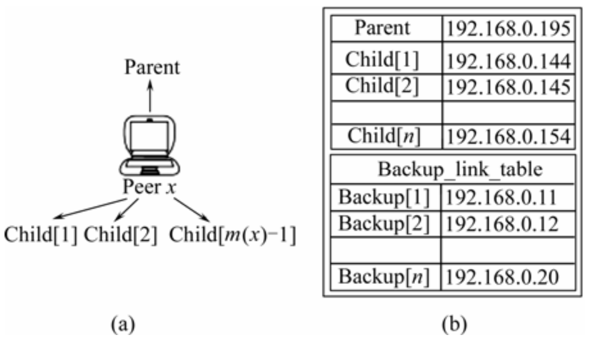
1.2 Assumptions and Data Structure

The P2P system model comprises of a large number of peers. Every peer shares some resources it holds and contributes parts of its resources for playing a part in the P2P overlay network. We assume that each peer knows an entry point when it first joins the system.

After getting the information of active peers, new peers can establish their own routing tables according to those in the active peers. However, in our system, host_cache is used to maintain the current root of each tree as an entry point for new peers to join the tree.

Furthermore, we assume each peer x will calculate a value (i.e. $c(x)$) to express its capacity when it joins the system. We will define the capacity in this thesis as the computing power and network bandwidth of peers. The $c(x)$ of each peer will be used in the join process to decide how many peers it can accept as children initially. We define $m_i(\mathrm{x})$ for a peer as the number of overlay links used in the ith tree. Therefore, if there are k overlay trees in the system, we can obtain that the sum of $m_i(x)$ is less than $c(x)$. We use $d_i(x)$ to express the overlay links in use in the ith tree of a peer. For the succinctness, we abbreviate them to $m(x)$ and $d(x)$ when we just discuss for one overlay tree. Figure 1 (a) shows overlay links maintained by a peer.

In addition, each peer keeps a routing table of neighbors (i.e. parent and children) and some backup links. Figure 1 (b) shows the format of the routing table in each peer. In general, tree is a structure with good scalability but difficult to be maintained because of its weak connectivity. We increase the reliability of our system by using the backup_link_table to maintain the overlay tree structures from partition.



Parent	192.168.0.195
Child[1]	192.168.0.144
Child[2]	192.168.0.145
Child[n]	192.168.0.154
Backup_link_table	
Backup[1]	192.168.0.11
Backup[2]	192.168.0.12
Backup[n]	192.168.0.20

Fig.1 The logical structure of overlay links held by a peer (a) and the routing table of a peer (b)

1.3 Basic Algorithms

1.3.1 Way peer join in

When a new peer ρ intends to join the system, it will ask the host_cache for the information of the current root of the overlay tree that peer ρ would like to join. It then uses the information to communicate with the root γ. Algorithm 1 depicts a new peer joining.

Algorithm 1

```
Join (tree $T$ ) {
// peer $\rho$ connects to host_cache and gets the information of the root of tree $T$
$\gamma$ = host_cache( $T$ )
if $d(\gamma) < m(\gamma)$
    then Graft ( $\rho$, $\gamma$)
    else
        $\varphi$ = Sub_tree_root($\gamma$)
        Graft ($\rho$, $\varphi$ ) }
Graft (peer $\mu$, peer $v$) {
$\mu$.parent =$v$
find an $\varepsilon$ such that $n$.child[$\varepsilon$ ] is NULL
$v$.child[$\varepsilon$ ] =$\mu$
Height_check($\mu$, $v$); }
Sub_tree_root (peer $\omega$) {
    for $i \leftarrow 1$ to $d(\omega)$
    do find $\omega$.child[$i$] has a maximum value, $f$=$m$ ($\omega$.child[$i$])−$d$($\omega$.child[$i$])
    if $f > 0$    then return $\omega$.child[$i$]
    else choose a random number $\delta$ between 1 and $d(\omega)$
    return $\omega$.child[$\delta$] }
Height_check (peer $\alpha$, peer $\beta$) {
if height($\beta$) $\leq$ height($\alpha$)
    then height($\beta$) = height($\alpha$) + 1
    $\alpha$=$\beta$ and $\beta$=$\beta$.parent
    while $\beta \neq$ NULL
Height_check($\alpha$, $\beta$) }
```

1.3.2 Tree maintenance

In our system, there are multiple tree-like structures in the overlay network. As we know, if a link of the tree structure is failed or broken, the tree will be partitioned into multiple sections. This causes a big problem for our system, because we need to gather all peers with similar interests and resources to a single overlay tree. Therefore, it is necessary to keep the routing table of all peers accurate in our system. We solve it by sending periodical messages (i.e. PING) to parent and children to check the states of them.

Each peer keeps a backup_link_table to hold some ancestors as the reserved parent. If a peer detects that its parent is off-line or has no response, it will choose one backup peer in this table and try to connect to it as a child. Choosing the new parent is given in Algorithm 2. There are many ways to decide how many and which peers to be put into the backup_link_table. In our system, the size of the backup_link_table is setup by the system in advance. Peers that are contacted by peer ρ in its join process will be put into its backup_link_table.

Algorithm 2

```
Recover (peer $\rho$) {
if there is an active peer $\omega$ in the backup_link_table and height($\omega$) is the
        closest to height($\rho$)
if $d(\omega) < m(\omega)$
    then Graft($\rho$, $\omega$) // Algorithm 1
    else $\kappa$ = sub_tree_root($\omega$)
    Graft($\rho$, $\kappa$)
else Join( $T$ ) }
```

1.3.3 Exploiting the heterogeneity of peers

The join process of a peer and the delivery of a query are all first handled by the root of the corresponding overlay tree. Queries are passed in a top-down fashion from the root to the leaves according to their TTL value. The Algorithm 3 shows how to moving powerful peers closer to the root.

After finding a candidate parent γ in the join process, a peer ρ will compare its capacity (i.e. c (ρ)) with c (γ) of the candidate parent. If the capacity of peer ρ is higher than γ, it will exchange its position with γ in the overlay tree.

```
    Algorithm 3
    Replace (peer $\rho$, peer $\gamma$) { //$\gamma$ is ρ.parent
  if $\gamma \neq$ NULL and $c(\rho) \leqslant c(\gamma)$
      then break
      ele lock the data structure of peer $\rho$ and $\gamma$
  for $i \leftarrow 1$ to $d(\gamma)$
      do $\gamma$.child[$i$].parent = $\rho$
   $\rho$.parent = $\gamma$.parent
   height($\rho$) = height($\gamma$)
   $\gamma$.parent = $\rho$
   Height_check($\gamma$, $\rho$)
   Replace ($\rho$, $\rho$.parent) }
```

1.3.4 Query delivery process

The Algorithm 4 shows the method of forwarding queries to a large amount of peers in an overlay tree for resource discovery.

Wherever a query is issued, it will be first forwarded to the root peer and passed down to the leaves. Note that the termination of a query depends on its TTL which is given by the query issuer. The TTL value will be decreased by one when the query travels across one hop in the overlay.

```
    Algorithm 4
    Route (query $\varphi$) { // for peer $\rho$
    if TTL of $\varphi > 0$
        then decrease TTL of $\varphi$ by 1
            for $i \leftarrow 1$ to $d(\rho)$
                //$\rho$ will pass query $\varphi$ to all its children
                do deliver $\varphi$ to $\rho$.child[$i$]
    else break }
```

1.4 Dynamic Exchange Methodology

Consider the example in which there is an overlay tree for searching and downloading document files. Figure 2 (a) is the ideal topology. After passing a large number of queries and matched files for a while, the load status of the peers may be much different. Figure 2 (b) shows the load change of the peers after a period of time.

We utilize the PING and PONG messages to check if the neighbors are active or not. We attach the status information of the peers to their PING packets that will be periodically passed to their parent. The Algorithm 5 presents a way of dynamically adjusting the positions of two peers. After gathering all PING messages from the children, a peer ρ will compare the information with its own. If peer ρ detects that some peers are better than it, ρ will exchange its position with the best child.

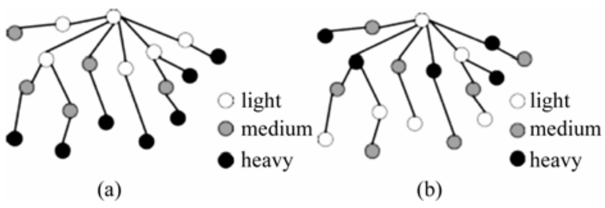


Fig.2 Effects of dynamic state changes

(a) The ideal tree in our system; (b) The load changes of peers after a while

```
    Algorithm 5
    Monitor ( peer $\rho$) {
    for $i \leftarrow 1$ to $d(\rho)$ // $\Phi$ is the metric
        do receive PING packets from $\rho$.child[$i$], extract
the specific item $\Phi_i$
    from the packet
    if $\Phi_i > \Phi_\rho$
       then put $\Phi_i$ into exchange_table
          while exchange_table is not empty
              do find the maximum $\Phi_i$ and return $i$
    Replace ( $i$, $\rho$) }
```

2 Experiments and Simulation

2.1 Evaluation with Real System

We implemented our system using Java JDK v1.4.2. For building a real test environment, we ran our system in the PC cluster of our lib. There are 64 machines interconnected with 100 Mb/s Ethernet. Each of them has a 2 GHz AMD ThunderBird MP2000 CPU and two 512 MB DDR RAM. The operating system is Linux with kernel v2.4.19.We selected 20 active machines as new peers to join in our overlay system (see Table 1). Each of them held a neighbor table with a size different from 1 to 8 and a backup_link_table with 5 entries. In addition, a peer issued a PING message to its parent node every 10 seconds.

Table 1 Settings of simulation

Rank	Rate/%	Capacity	Failure rate/%	Size of backup link table
R_1	20	2	81	1
R_2	45	4	27	3
R_3	30	8	9	5
R_4	4	6	3	7
R_5	1	32	1	9

We used two different join models of peers to get the different cost of the join process. The joining model of 20 peers was in a sequential way. We measured the total time of each peer from the time the program is started until an active parent node is obtained.

The first model is that peers join the system from the strongest to the weakest ones. Figure 3(a) shows the elapse time in milliseconds against the number of nodes joining the system. We note that when the 10th peer joined the system, it must start a sub_tree_root process to get a peer in Level-1 (the Level of the root is 0) as the parent because the root had reached its limit of overlay connections.

The second model of peer joining is from the weakest to the strongest ones. Figure 3(b) shows the results. In Fig.3 we can see the real statistics obtained by experiments are almost the same with the estimated values.

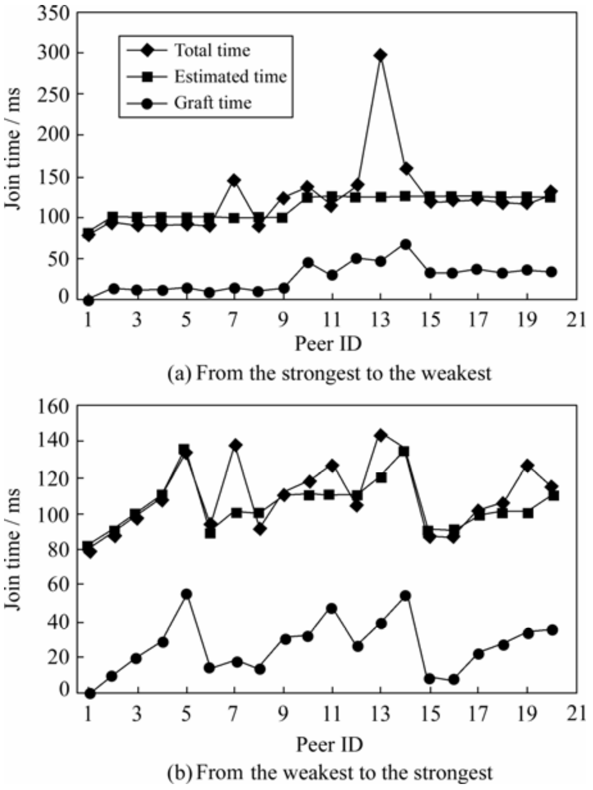


Fig.3 Elapse time of peer joining

2.2 Simulations

We developed a simulator to simulate a large number of peers (up to 50 000) typical of a real worldwide P2P environment[11-14]. We will show that our system can adapt to different needs of high level applications and adjust the structure to best meet them. We summarize the settings of the peers in Fig.3. We classified system nodes into five ranks that are expressed as R_1, R_2, R_3, R_4 and R_5. The failure rate of each peer represents the probability of a node failing per simulated day. In each simulated day, there will be 20% of peers leaving the system and then immediately rejoining the system to maintain the system size and heterogeneous percentage.

We can see from the Fig.4 that the average loading of peers at each level drops down by using the dynamic replacement. From the other simulations with different network sizes (such as system size = 30 000 and 50 000), we can obtain the similar result, the dynamic exchanging scheme can achieve better load balance for our system.

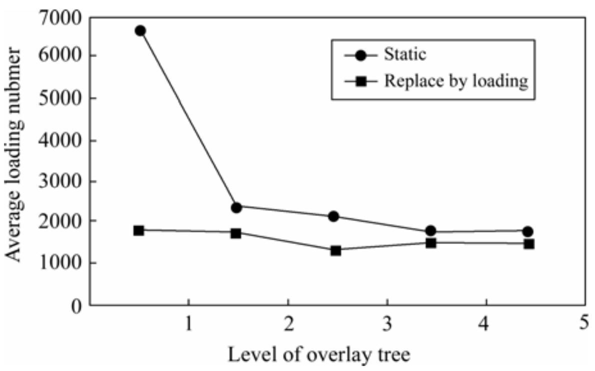


Fig.4 Loading status against the level of overlay tree with system size=10 000

3 Conclusion

We presented a multiple-tree-like overlay system for resource discovery in unstructured P2P networks. Forwarding of queries in our system is along the tree, which can eliminate redundant messages. Unlike previous works, peers in our system do not need to pass queries to unrelated peers. Moreover, we exploit the capacity of peers and utilize more powerful peers to handle more jobs. By dynamically adapting the tree structure, our system can serve as an application-specific overlay network to meet different needs of applications. And the simulations show that our system can adapt to meet different needs of applications with small costs.

Further studies we will design a more efficient and reliable system, and trust management should be considered in system.

References

[1] Jiang Song, Zhang Xiaodong. FloodTrail: An Efficient File Search Technique in Unstructured Peer-to-Peer Systems[C]// *Proceedings of* 2003 *IEEE Globecom Conference.* California: IEEE Press, 2003: 2891-2895.

[2] Gnutella[EB/OL].[2005-12-10]. *http://rfc-gnutella.sourceforge.net.*

[3] Ratnasamy S, Francis P, Handley M, *et al*. A Scalable Content-Addressable Network[C]// *Proceedings of ACM SIGCOMM* 2001. New York: ACM Press, 2001:161–172.

[4] Hsiao H C,King C T. Tornado: A Capability-Aware Peer-to-Peer Storage Overlay [J]. *Journal of Parallel and Distributed Computing*, 2004, **64**(6): 747-758.

[5] Clarke I, Sandberg O, Wiley B, *et al*. A Distributed Anonymous Information Storage and Retrieval System [C] //*Proceedings of International Workshop on Design Issues in Anonymity and Unobservability*. *LNCS*, 2001, **2009**: 46-66.

[6] Anirban M, Yi L, Masaru K. On Improving the Performance Dependability of Unstructured P2P Systems via Replication [EB/OL].[2005-11-20]. *http://www.tkl.iis.u-tokyo.ac.jp/ Kilab /Research/Paper/*2004*/Anirban-DEXA*-200409*.pdf.*

[7] Stoica I, Morris R, Karger D, *et al*. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications[C]// *Proceedings of ACM SIGCOMM* 2001. New York: ACM Press, 2001:149-160.

[8] Jiang Song, Guo Lei, Zhang Xiaodong. LightFlood: An Efficient Flooding Scheme for File Search in Unstructured Peer-to-Peer Systems[C]//*Proceedings of the ICPP*'03. California: IEEE Press, 2003: 627-635.

[9] Yang Beverly, Patrick V, Hector C M. Evaluating GUESS and Non-Forwarding Peer-to-Peer Search [EB/OL]. [2005-12-20].*http://www-db.stanford.edu/~byang/pubs/guess.pdf.*

[10] Jagadish H V, Beng Chin Ooi. BATON: A Balanced Tree Structure for Peer-to-Peer Networks[C]//*Proceedings of the* 31*st VLDB Conference*. Norway: ACM Press, 2005:661-672.

[11] Acosta W, Chandra S. Unstructured Peer-to- Peer Networks-Next Generation of Performance and Reliability [EB/OL].[2005-11-10]. *http://dawn.cs.umbc.edu /INFOCOM* 2005 */acosta-abs.pdf* .

[12] Anirban Mondal, Yi Lifu, Masaru Kitsuregawa. On Improving the Performance Dependability of Unstructured P2P Systems via Replication[EB/OL].[2005-11-20]. *http://www.tkl.iis.u-tokyo.ac.jp/Kilab/Research/Paper/*2004 */Anirban-DEXA*-200409*. pdf* .

[13] Berfield A, Qu Huiming. Node Mobility in Unstructured P2P Networks[EB/OL].[2005-11-20]. *http:// www.cs.pitt.edu/~ andreea/publications/mobility-p*2*p.pdf* .

[14] Castro M, Druschel P, Kermarrec A-M *et al*. Splitstream: High-Bandwidth Multicast in Cooperative Environments[C]// *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*. New York: ACM Press, 2003:298-313.

□