A Content-Based Resource Location Mechanism in PeerIS

Bo Ling[1], Zhiguo Lu[1], Wee Siong Ng[2], BengChin Ooi[2], Kian-Lee Tan[2], Aoying Zhou[1]
[1]Department of Computer Science and Engineering
Fudan University
Shanghai, China, 200433
{lingbo,luzhiguo,ayzhou}@fudan.edu.cn

[2]Department of Computer Science
National University of Singapore
3 Science Drive 2, Singapore 117543
{ngws,ooibc,tankl}@comp.nus.edu.sg

Abstract

With the flurry of research on P2P computing, many P2P technical challenges have emerged, one of which is how to efficiently locate desired resources. Advances have been made in this hot research field, where the pioneers are Pastry, CAN, Chord, and Tapestry. By using the functionality of distributed hash table, they have achieved fair effectiveness. However, they have many common limitations, such as ignoring the autonomous nature of peers, and just supporting weakly semantic functions. In this paper, according to the reality in the distributed network, we propose a content-based location mechanism, which not only keep the autonomy of peers, but also support approximate query and finer granularity of content sharing. Furthermore, this mechanism also facilitates P2P system to evolve dynamically. We have also used PeerIS, a P2P based information system used to verify it and obtained satisfactory results.

1. Introduction

A long-standing tenet of distributed systems is that the strength of a distributed system can grow with more hosts participating in it [6], since each participant contributes data and computing resources to the overall system, the wealth of the community can scale with the number of participants. A peer-to-peer (P2P) distributed system is one in which all participating computers (which can be called nodes or peers) have equal capabilities and responsibilities and can interact with each other directly and symmetrically. Instead of strictly decomposing the system into clients (who consume services) and servers (who provide services), nodes can decide to be clients as well as servers. This grand architecture has many potential advantages, e.g., scalability, robustness, lacking of need for administration, and even anonymity and resistance to censorship. Thus, this computing model is now viewed as a potential technology that could re-architecture the distributed architectures.

Unfortunately, the recent flurry of research has quickly shown the immense technical challenges associated with P2P computing, particularly in term of scale, one of which is how to efficiently locate desired resources in this kind of decentralized, ad-hoc and dynamic networks, because this decides whether their resources can be efficiently used or not. Furthermore, this challenge greatly affects the salability of P2P systems and the realization of their other potential advantages. Although, CAN [10], Pastry [1], Chord [13], and Tapestry [16] have focused on this research field and made some progress. However, in these systems, files are associated with a key (typically produced by hashing the file name) and each peer in the system is responsible for storing a certain range of the keys, regardless of which user the corresponding files belong to. Obviously, this method has many limitations. Firstly, it ignores one of the most important natures of P2P nodes–autonomy, which is against the idea of P2P. Secondly, it just supports precise query but not approximate query. And thirdly, it just supports file-level sharing. In this paper, we propose a content-based resources location mechanism, which can solve all of the problems mentioned above. Our automatic mechanism is based on the reality in distributed networks, e.g., the Internet, where users almost just maintain and search for their favorite materials. Thus, their favorites can be captured according to the category distribution of the files that they maintain. Furthermore, their query distribution bares certain relation with that of their files. With the strategy of file

classification, our mechanism can easily determine the similarity degree of favorites between peers as well as predict the category distribution of their queries, and further guarantee the peers with high similarity of favorites clustered closely, so that efficient location can be achieved. Besides supporting autonomous nature, finer granularity of sharing and approximate query, a sophisticated tactic in this mechanism facilitates P2P systems to evolve dynamically. We have implemented this mechanism in PeerIS, a P2P-based information system built upon the BestPeer [14, 8], and tested it. The experiment results further verify the effectiveness of our mechanism. In short, the main contributions we make in this paper are as follows:

- We define the similarity degree of favorites and behavior between peers and further present their computing formulae.
- We propose a content-based resource location mechanism and its implementation procedure, which not only maintains the autonomous nature of peers but also supports approximate query and fine granularity of content sharing. Furthermore, a sophisticated tactic in this mechanism facilitates the system to evolve dynamically.
- We have implemented our mechanism in PeerIS and tested it. Our experiment results further verify its effectiveness.

The rest of the paper is organized as follows: in Section 2, we describe the architecture of PeerIS, a materialization of our resource location mechanism; Section 3 details the procedure of the implementation of our mechanism; Section 4 describes the mechanism of query processing while Section 5 we report experiment results and analysis; we discuss the related work in Section 6 and conclude in Section 7.

2. PeerIS

2.1. Architecture of PeerIS

We have implemented our content-based resource location mechanism in PeerIS: a prototype of distributed information system built on the top of BestPeer [14, 8]. BestPeer is a generic P2P system designed to serve as a platform on which P2P applications can be developed easily and efficiently, which has several features that distinguish itself from other existing P2P systems. Firstly, it is the first system having integrated two powerful technologies: mobile agent and P2P technologies. Secondly, it can shares finer granularity of content and computational power. Thirdly, its peer can dynamically reconfigure itself by keeping peers that benefit it most. Finally, BestPeer introduces distributed LIGLO naming services.

Since PeerIS is a prototype built upon BestPeer with extending many functions, its architecture is very similar to that of BestPeer. The network also comprises two kinds of entities, i.e., several LIGLOs, who connect and interact with each other in P2P manner, and a large number of peers. Each LIGLO connects with all of its peers while a peer can just connect with a limited number of other peers (due to its computing and bandwidth capability) at a given time. In PeerIS, the relationship among peers can define three types: (a) ***confidants***: two peers connecting directly; (b) ***acquaintance***: two peers interacting with one another via *confidants* and; (c) ***stranger***: two peers without any inaction. Each peer manages its *confidants* and *acquaintances* with a *ConfidantCircle* and *AcquaintanceCircle* respectively. Our location mechanism facilitates each peer to maximize the number of its confidants with highest similarity of favorites and behavior under the *constraints* of its capability. Thus, in PeerIS, peers with different favorites and behavior form different clusters, an in each cluster, the more similar they are, the nearer (the less logical hops between) two peers are.

Compared with BestPeer, a LIGLO in PeerIS is improved in the following aspects. Firstly, it can monitor the distribution of file categories in some peers as well as its change trend. Secondly, according to their metadata or application tables, it can recommend appropriate *confidant* candidates to applicants. Thirdly, a new LIGLO can dynamically select and replace "super" nodes among peers according to their (a) computing resources and bandwidth capabilities, (b) volume of files and their category distribution and, (c) the behavior of being seldom offline. All of the natures mentioned above are helpful for applicants to bootstrap themselves. Finally, instead of checking the status of its peers with uniform frequency, a new LIGLO can treat different nodes with different policies, i.e., it pays more attention to super nodes while pays less attention to common nodes, which can make computing and bandwidth resources efficiently used.

Each peer in PeerIS has four components that are loosely integrated, whose architecture is shown in Figure 1.

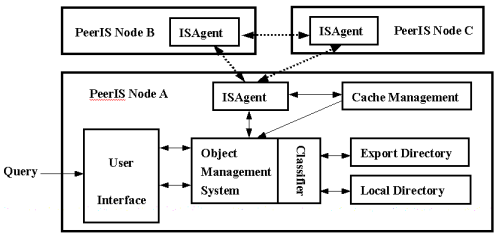


Figure 1. The Architecture of PeerIS Peer

The first component is an object management system,

which is responsible for file processing (such as eliminating stop words and doing stemming), extracting feature terms and classifying, storing, and retrieving of the files in the peer. Thus, the system can be used on its own as an autonomous Information System (IS) when out of PeerIS. For each file it maintains, the associated metadata (index terms, category, author etc.) stored in a ***Local Directory***, which has a logical ***Export Directory*** to manage the metadata of its sharable files.

Our object management system integrates a file classifier, which adopts k-Nearest-Neighbor (kNN) method. According to [15], kNN classifier has many properties that make it suitable to classify peer's content to achieve our goal. To support kNN strategy, the files are presented with a vector space model (VSM) [12]. Furthermore, the model can harmoniously support query processing, since its intermediate results can be efficiently reused when managing files. With the application of kNN classifier, the category distribution of peers' files and further favorite similarity between peers can be determined.

The second component is an Agent system called ISAgent. ISAgent provides the environment for mobile agents to operate on. Each PeerIS node has a *Master* agent, which clones and dispatches *Worker* agents to its relative neighboring peers and manipulates answers. It also maintains the statistics obtained from the query processing and manipulates the reconfiguration of its *confidants*. The third component is a cache manager, which facilitates caching remote data in secondary storages, and determines the caching/replacement policy. The last component is the user interface, which provides a user-friendly environment for users to submit their queries, maintain their sharable files, and insert as well as delete files.

2.2 Criterion

Each peer in a P2P network has three sorts of fundamental resources, i.e., computing resources, information or data, and bandwidth [5], which are used to depict the characters of peers naturally:

1. The computing resources of a peer, say P_i, are quantified by $CPU(P_i)$, $Memo(P_i)$ and $storage(P_i)$ respectively. As for the storage capability of a peer, it just refers to the part of storage used to store sharable data.

2. Each peer, say Pi, just has a limited volume of bandwidth. Since each connection consumes amount of bandwidth, the number of connections that peer P_i can maintain has a upper bound presented by $MaxConn(P_i)$ and its connections have a real time value: $ConnRt(P_i)$, obviously $ConnRt(P_i) \leq MaxConn(P_i)$, which is the basis of maintaining and adjusting a peer's *confidants*.

3. Each peer maintains a collection of files and shares part of them with others. Since, in distributed networks, such as the Internet, individual users just maintain their favorite materials as well as search their favorite materials. Based on it, we can assume that in most circumstances, the category distribution of queries submitted by rational users bares similarity to that of the files they maintain. According to this rationale, we employ the file classification to determine the users' favorites and determine their similarity and further predict the distribution of their queries. Thus, our mechanism can guarantee that each peer in PeerIS just maintains the ones that share similar favorites (which can also be presented with Group_IDs) as its *confidants*, so that answers to a given query can be efficiently located in a small scope in most circumstances.

As follows, we detail the important criteria used in our mechanism and their computing formulae.

- Similarity degree of favorites. This is the precondition for a peer, say P_i, to determine whether another peer, say P_j, can be its *confidant* or *confidant candidate*. It is captured by the category distribution of their sharable files, which is obtained by the following formula:

$$SimFavorites(P_i, P_j) = \frac{\vec{D}(P_i) \bullet \vec{D}(P_j)}{\left|\vec{D}(P_i)\right| \times \left|\vec{D}(P_j)\right|} = \frac{\sum_{t=1}^{c} w_{t,i} \times w_{t,j}}{\sqrt{\sum_{t=1}^{c} w_{t,i}^2} \quad \times \quad \sqrt{\sum_{t=1}^{c} w_{t,j}^2}}$$

  Above, $\vec{D}(P_i)$ and $\vec{D}(P_j)$ are the vector of P_i's and P_j's file categories respectively, while $\left|\vec{D}(P_i)\right|$ and $\left|\vec{D}(P_j)\right|$ are their norms. Further, $w_{t,i}$ and $w_{t,j}$ are the weight of file category t, which peer P_i and peer P_j hold respectively. For simplicity, we can use the following formula to calculate the weight of category t of files in the peer P_i:

$$w_{t,i} = \frac{n_{t,i}}{N_i}$$

  Above, $n_{t,i}$ is the number of file of category *t* which peer P_i holds, while N_i stands for the total number of files that peer P_i offers to share with other peers.

  With the criterion, a peer can easily determine its own *confidants* and further optimize its *confidants* by adjusting the threshold of similarity. Indeed, it is the basis of our automatic resource location mechanism.

- $Behavior(P_i)$ and $SimBh(P_i, P_j)$. Here, $Behavior(P_i)$ stands for the behavior of peer P_i, which indicates the fact when peer P_i is on the P2P network and interacts with other peers, which can be obtained from its system log by agent. $SimBh(P_i, P_j)$)is a metric of the similarity of behavior between P_i and P_j which can be computed by the following formula:

$$SimBh(P_i, P_j) = \frac{(Time(P_i) \cap Time(P_j))}{24}$$

stands for the duration when both peer P_i and peer P_j are on the P2P network and connect with each other during a day, while 24 mean there are 24 hours in a day. Thus, this leads to the constraint: $0 \leq SimBh(P_i, P_j) \leq 1$. In practice, it is an important metric for two peers to maintain their *confidant* relationship, since if P_j is not on the network while its *confidant* P_i is, then P_j maybe be replaced by other peers, and the less time P_j shares with P_i, the higher probability that P_j is replaced by others. This metric is very useful for a peer to optimize its *confidants*.

- $Rim(\ P_j \ \ to \ \ P_i)$. It is a metric of *relative importance* of peer P_j to peer P_i. And can be calculated by the following formula: $Rim(\ P_j \ \ to \ \ P_i) = \alpha * \left(\sum_{Sum} \frac{QueryHits(P_j)}{Hops(P_j)}\right)$
$+\beta * SimBh(P_i, P_j)$
$+\gamma * \sum_{Relative} ConnRt(P_j)$
$+\eta CPU(P_j) + \lambda Memo(P_j) + \mu Storage(P_j)$
Above, $\alpha, \beta, \gamma, \eta, \lambda$ and μ are coefficients and $\sum_{Sum} \frac{QueryHits(P_j)}{Hops(P_j)}$ is the sum of *Benefit-Cost* of *QueryHits* of peer P_j, which is given by:

$$\sum_{Sum} \frac{QueryHits(P_j)}{Hops(P_j)} = \frac{\sum_{Local} QueryHits(P_j)}{Hops(P_j)}$$

$$+ \sum_{A:Acquaintance} \frac{\sum QueryHits(P_A)}{Hops(P_A)}$$

Above, $\frac{\sum_{Local} QueryHits(P_j)}{Hops(P_j)}$ stands for total *Benefi-Cost* of *QueryHits* obtained by peer P_j from its own file collection, while $\sum_{A:Acquaintance} \frac{\sum QueryHits(P_A)}{Hops(P_A)}$ is the sum of *Benefit-Cost* of *QueryHits* obtained by peer P_j via its *confidants* and *acquaintances*; while the last three components stand for the contributions of P_j's computing capability; Especially, $\sum_{Relative} ConnRt(P_j)$ is the sum of peer P_j's current connections with the peers who reply P_i with *QueryHits*, which is an important factor to determine P_j's relative importance.

With comprehensive metric, a peer can correctly adjust its *confidants*, so that P2P systems can dynamically evolve, which is one of predominant features of our automatic location mechanism.

2.3 Communication Mechanism of PeerIS

There are two kinds of communication manners in PeerIS, i.e., pure message and mobile agent, to cater to different needs. Pure message in PeerIS, which is generated by a peer periodically just as heartbeat, is used to probe the status of other peers. This manner is very similar to that of Gnutella [4].

The second communication manner in PeerIS is mobile agent. Mobile agents that move in PeerIS also maintain most of the information in the pure massage. However, its "Workloads" are different. Here, we briefly describe Workload-Descriptor conveyed by agents:

- *Connection*. It is used when a peer applies a new connection with another. This mechanism is very useful during course of initialization of a new peer and when a peer adjusts its *confidants*.

- *Connection_Proved*. It is the reply of *Connection*. When a peer receives a *Connection* request conveyed by a mobile agent, it extracts its constraints from the agent, and computes the similarity or relative importance between the requester and itself, or the relative importance of the requester. If the computing results satisfy its metrics, it accepts the application and replies the requester with a *Connection_Proved*.

- *Query*. It is the fundamental mechanism to realize file searching and sharing. Along with it, mobile agent carries the operational constraints, such as category of query, key words, and so on. If the receiver holds the answer to the query, it replies with a *QueryHits*.

- *QueryHits*. It is the reply of *Query*. Together with *QueryHits*, agents carry relative information on the reply, such as BPID and IP address of holders' and the query routing path. The former is used to satisfy the current query, while the latter is used to enable the recipient to optimize its *ConfidantCircle*.

3. Implementation Procedure

3.1 Initiation

After the PeerIS software is installed, it is firstly used by the new peer to process the files it maintains, so that the relative metadata are obtained. By now, the peer is an autonomous information system, if it dose not join the PeerIS

network. The following process is taken by a peer to become a participant of PeerIS:

1. The user should fill an application table, which reflects his/her favorites. Together with the metadata of the files the peer maintains, the user registers with a LIGLO with the above information, which is similar to registering a mail server in the Internet environment.

2. The LIGLO server will issue the peer with a globally unique identifier, i.e, BPID, which takes form of a (LIGLOID, NodeID) pair, where LIGLOID is the IP address of the LIGLO server, while NodeID is the unique ID for the peer assigned by the LIGLO. With this structure, the BPID can serve to uniquely recognize this peer regardless of its IP address changing.

3. At the same time, the LIGLO computes the similarity between the applicant and some of the super nodes by using their metadata. According to result, the LIGLO recommends the applicant a list of appropriate super nodes as the applicant's *confidant* candidates.

4. The applicant peer's *Master* agent clones several *Worker* agents (who carry *Connection* request, metadata and the application table), and dispatches them to each of the recommended super nodes (*confidant* candidates). On receiving a *Worker* agent, the super node, say P_i, firstly makes sure that it has not been visited by the agent and makes its TTL-1. Then, P_i computes their similarity, and decides whether to receive it as a *confidant*. If the result is satisfactory and $ConnRt(P_i) < MaxConn(P_i)$, it replies the applicant with a *Connection_Proved* together with the computing result to the applicant (say P_j); Or if $ConnRt(P_i) = MaxConn(P_i)$, the super node P_i would logically delete its least important confidant to accept the request. Further, P_i updates its *ConfidantCircle* if P_j issue P_i an Active message in a predefined period (for conforming). Else, if a super node regards the applicant is not appropriate enough, it turns down the *Connection* request and recommend the applicants some other super node (if possible) if TTL$>$0. Indeed, in order to guarantee a new peer to be able to bootstrap itself successfully and rapidly, our mechanism offers a higher priority to the applicant.

5. Since the *Connection_Proved* messages from different super nodes arrive at the applicant (say P_j) at different time, the applicant firstly stores them in its buffer firstly. After a predefined period, it ranks their similarity and chooses the best ones to be its confidants and then sends an *Active-Ok* message to each of the selected candidates. Then, the new peer builds up its *ConfidantCircle*. By now, the initiation process is finished and the new peer has become a participant of the PeerIS.

3.2 Maintenance

Because of the dynamic and ad hoc nature of peers, a peer (say Pi) in the P2P network must continually maintains its confidants, which covers the following aspects:

- Similarly to heartbeat, P_i periodically issues an *Active* message to each of the peers in its *ConfidantCircle* (say P_j,) on the condition that P_i does not receives that from P_j. If P_i receives an *Active_Ok* from P_j then P_i regards P_j is active, or there are three possible situations of its confidants (P_j), i.e., *InActive*, *Active but IP changed*, and *Active but confidant relation closed*. According to different situation, P_i takes different actions. We delay the discussion in Subsection 3.3.

- Dealing with *Connection* requests, each active peer, especially super node will definitely confront the *Connection* requests. In PeerIS, to solve the issue, the active peer P_i first compute their similarity of favorites and behavior or the relative importance of the requester, and decides to accept the request and forwards to its *confidants* or just forwards to its appropriates *confidants* or just drops it.

- If unfortunate, all of its *confidants* are not able to connect, then Pi must issue the *Connection* requests to its *acquaintances* according to its query history, or else, it has to bootstrap itself with the help of its LIGLO.

3.3 Rejoining of Peer

A peer P_i who has been a participant wants to rejoin the PeerIS network after disconnection or failure, taking the following process:

- Peer P_i sends its current IP address to its LIGLO firstly to allow its LIGLO to update its IP address if it has changed;

- Secondly, it sends an *Active* message to each of peers in its *ConfidantCircle* (say P_j) to restore its connections with them. If P_i receives an *Active_Ok* message from P_j, then it restores their connection successfully. Otherwise, with the help of LIGLO, P_i should firstly determine the status of the *confidant*, i.e., InActive, Active but IP changed, and Active but confidant relation closed, then it takes corresponding actions. For the first situation, it just maintains P_j in its *ConfidantCircle* for a certain period, which is decided by P_j's relative importance to it. For the second situation, P_i

can track to P_j's LIGLO and obtains P_j's current IP address, so that it can restore their connection. For the last situation, P_i just reissues a *Connection* request to P_j, if P_i receives a *Connection_Proved* message, then their *confidant* relationship is restored, else, P_i can replace P_j if it confronts appropriate peers.

- By now, peer Pi has finished its rejoining process.

4 Query Processing

4.1 Mechanism of Query Processing

In order to obtain the goal of approximate query and content granularity of sharing, the query processing in PeerIS is supported by vector space model [12], which is also harmonious with the mechanism of file classification described above. In vector space model, the index terms in both files and query are non-binary weights, which are ultimately used to compute the similarity between files maintained by peers and the submitted query. Furthermore, the retrieved files are ranked in decreasing order of the degree of similarity.

Definition 4.1 *For the vector model, the weight $w_{i,j}$ associated with a $pair(k_i, d_j)$ is positive and non-binary, where k_i and d_j are the ith index term and jth file respectively. Further, the index terms in the query are also weighted. Let $w_{i,q}$ be the weight associated $pair(k_i, q)$ where $(k_i, q) \geq 0$. Then the* query vector *$\vec{q}$ is defined as $\vec{q} = (w_{1,q}, w_{2,q}, ..., w_{t,q})$ where t is the total number index term in the relative peers. As before, the* file vector *for the file d_j is presented by: $\vec{d} = (w_{1,j}, w_{2,j}, ..., w_{t,j})$ Thus, the similarity between the q and d_j can be computed by*

$$Sim(d_j, q) = \frac{\vec{d_j} \bullet \vec{q}}{|\vec{d_j}| \times |\vec{q}|} = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,q}^2}}$$

Above, $\vec{d_j}$ and $\vec{q}$ are the vector of file d_j and query q respectively, while $\left|\vec{d_j}\right|$ and $|\vec{q}|$ are their norms. In order to determine the weights in the above formula, we give out the following definition and relative formulae.

Definition 4.2 *Let N be the total number of files in a peer and n_i be the number of files where the index term k_i appears. Let $freq_{i,j}$ be the* raw frequency *of term k_i in the file d_j. Then the* normalized frequency *$f_{i,j}$ in file d_j is given by*

$$f_{i,j} = \frac{freq_{i,j}}{\max! freq_{l,j}}$$

Above, the $\max! freq_{l,j}$ stands for maximal value of the raw frequency *of all index terms, which are mentioned in the text of file d_j. If terms k_i does not appear in file d_j, then $f_{i,j} = 0$. Further, let idf_i be the* inverse file frequency *of k_i, given by $idf_i = \log \frac{N}{n_i}$. Thus, a weight is given by,*

$$w_{i,j} = f_{i,j} \times idf_i = \frac{freq_{i,j}}{\max! freq_{l,j}} \times \log \frac{N}{n_i}$$

4.2 Query Procedure

When a query is submitted, it is firstly parsed and classified by the requester peer, e.g., P_i locally. Then it is processed in a parallel manner, i.e., the query is processed locally as well as is conveyed by *Query* agents to relative *confidants* or *acquaintances*, who have files sharing the same category as the query. In practice, the process adopts a two-phase strategy when a peer access remote peers (*confidants* or *acquaintances*). In its first phase, *QueryHits* to the query from remote peers are returned to requester firstly, which are firstly ranked automatically and then selected by users. In the second phrase, the results of first stage are directed to the remote peers who hold the results and the answers are finally returned. This strategy has two advantages. First, it minimizes the information overload and better utilizes the bandwidth of the network, since all *QueryHits* are selected semantically. Second, relative information on the peers (e.g., the query path and hops, connection status and behavior) that issue the *QueryHits* is returned to the requester by *Query* agents to the *Master* agent, which are helpful for the requester to reconfigure its *ConfidantCircle*, so that the network can dynamically evolve. We detail both local and remote query processing respectively.

1. Local Processing:

   - The query is parsed firstly when submitted. Then it is applied on the local dictionary in term of its category and promising results are displayed to the user immediately. Moreover, the results are ranked by the *Master* agent according to their $Sim(d_j, q)$.
   - At the same time, the *Master* agent clones *Query* agents and dispatches them to all relative *confidants* or *acquaintances*, which maintain files whose category is the same as that of the query, and they further forward the query to *confidants*' *confidants*, and so on until TTL=0. Together with the query, *Query* agents also convey (a) IP address of the requester, and (b) TTL. The former facilitates remote peers to directly return *QueryHits* to the requester, while the latter determine the lifecycle of *Query* agents. During the course of routing, they record the query paths.

- The *Master* agent waits for *QueryHits* from remote peers. On receiving, it ranks and presents them to the user for selection. At the same time, all the statistics for reconfiguring are also recorded. Then selected results are conveyed to the holders at the beginning of the second phase.
- Once final answers are retrieved, the *Query* agents are dropped.

2. Remote processing:

- The peer has not been visited previously, then *Query* agent is accepted and reduce its TTL by one.
- The *Query* agent searches against the *Export Dictionary* of the remote peer, which is similar to local processing. And the results with $Sim(d_j, q) \geq threshold$, together with reconfiguring information are directly returned to the requester by the *Query* agents, which takes form of *QueryHits*.
- If TTL > 0, then the agent clones one or more *Query* agents and dispatches them to relative confidants, else, the agent is dropped.

Indeed, the two phrases are only logical, i.e., the procedure of selection and retrieval is parallel and progressive, which can reduce the user's waiting time.

4.3 Evolution of System

One predominant feature of our mechanism is that it facilitates peers in P2P systems to dynamically reconfigure their *ConfidantCircle*, so that P2P systems can continuously evolve. Periodically, each peer, say Pi, evaluates the relative importance of each peer (say P_j) in its *ConfidantCircle* and each acquaintance (say P_k) in its *ConfidantCircle*. For peer P_i, if its $ConnRt(P_i) < MaxConn$, then it adds the most relative important *acquaintances* into its *ConfidantCircle* until $ConnRt(P_i) = MaxConn(P_i)$; Else, P_i tries to replace its least important confidants with the most important *acquaintances* on condition of $Rim(\ P_k \ \ to \ \ P_i) \ > Rim(\ P_j \ \ to \ \ P_i)$. To do it, P_i firstly sends a *Connection* request to P_k and if P_i receives *Connection_Proved* from P_k, then P_i build a connection with P_k and update its *ConfidantCircle*. And if P_i does not receive a *Connection_Proved* form P_j in a predefined period, then P_i regards it cannot make a connection with P_j. The procedure is described by pseudo-code in Algorithm 1.

Algorithm 1: Evolving Algorithm for P_i

```
begin
    foreach peer in P_i's ConfidantCircle, say P_j, and each relative peer P_k in P_i's AcquaintanceCircle do
        Compute Rim( P_j to P_i) and Rim( P_k to P_i) ;
        while Max(Rim( P_k to P_i) )> Min(Rim( P_j to P_i) ) do
            if ConnRt(P_i) < MaxConn(P_i) then
                add P_k into P_i's ConfidantCircle
            else
                Replace P_j with P_k
            Delete P_k from AcquaintanceCircle;
    return Updated ConfidantCircle
end
```

5 Experiment Analysis

In this section, we report some of our experiment results. Just as in [14], we also have carried out comprehensive experiments and got promising results. Since their results are very similar, in this paper, we just reports some results of PeerIS vs. Gnutella, when the reconfiguration and approximate search functionalities of PeerIS having been turned off. Moreover. In addition, we also test how the query distribution affects the performance.

5.1 Experiment Setup

The experiment environment is made up of 32 PCs with Intel Pentium 500MHz processor and 64M of RAM, and all the PCs are running Windows 2000 operating system. Among them, one serves as a LIGLO server and the other 31 PCs serve as peers. Every peer maintains 1,000 files and each of the files is around 5KB. Further, 80% of the files that a peer maintains belong to four specific categories and each category randomly ranges from 10% through 50%, while the rest 20% of files are randomly selected.

The experiments are conducted when the machines and network are fully dedicated. Further, the topologies of both PeerIS and Gnutella are randomly generated.

5.2 Experimental Results and Analyses

5.2.1 Comparison of *Benefit-Cost* of QueryHits

We first compare the Benefit-Cost of QueryHits in PeerIS and Gnutella, which implies how much bandwidth a query consumes to obtain a QueryHits. A randomly generated

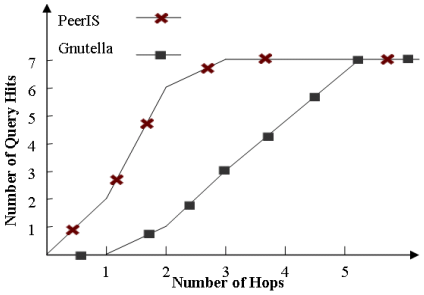


Figure 2. Benefit-cost of QueryHits

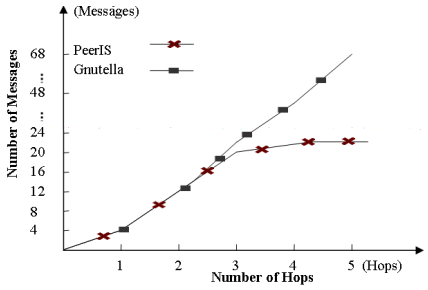


Figure 3. Bandwidth Consumption Comparison

query is submitted via a randomly selected peer (called base peer) to search one file belonging to the four favorite categories. The execution has been repeated ten times and the average results shown in the Figure 2.

In the figure, the base peers both systems have not got any *QueryHits* when hops is zero, which implies all of their replies are all obtained from remote peers, thus, they both carry out local processing and remote processing during the query processing. Further, the base peer in PeerIS obtains more *QueryHits* than that of Gnutella when their queries travel in small scope. Moreover, to obtain all answers, the PeerIS's base peer need only cover 3 hops while the Gnutella's base peer need cover 5 hops, because in PeerIS, our automatic mechanism guarantees peers with similar favorites clustered closely, its peers can obtain the relative replies in a smaller scope; while in Gnutella, confidants of each peer is randomly selected, answers to a query are randomly distributed among the network, a query has to traverse the whole system to get all available answers; Since our network just consists of 31 peers, they all can be covered within 5 hops.

5.3 Messages

The number of messages of a P2P system is determined by its location and routing mechanism, which implies the utility of bandwidth and further determines the scalability of the system. During the above experiment, we have also recorded the number of query messages at each hop in both systems respectively and plotted in Figure 3.

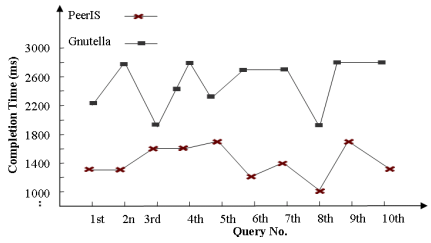
As shown in the figure, the number of messages in both systems is almost equivalent until the hops is around 3. However, after that point, the number of messages increases much faster in Guntella than in PeerIS. Since, in Gnutella, when a peer requests a file, it forwards the query to all of its *confidants* and its *confidants* relay the query to all their *confidants*, and so on, until the TTL=0, so that the number of messages in Gnutella increases exponentially. Although, the number of messages in PeerIS is almost the same as in Gnutella at the first hops, the reason is different. In PeerIS, peers with high similar favorites cluster closely and the nearer they are, the probability that they satisfy the queries of each other's is higher, so that the requester send the request to almost all of its *confidants*. However, with the hops increasing, the probability of *acquaintances* having the answers to the query decreases, and the farther the *acquaintance* is from the requester, the less probability it has answers to the query. Thus, there are less *acquaintances* needed to visit, as a consequence, the number of messages increases more and more slowly.

5.4 Rate of Query Completion

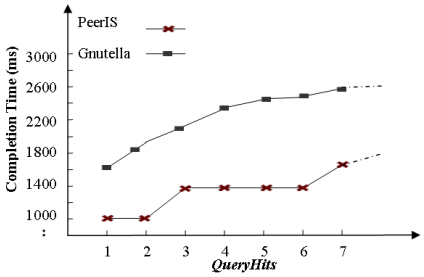
The rate, at which queries complete and *QueryHits* are turned, is one of the most important indexes of performance of P2P systems. In this experiment, the completion rates of different queries as well as returning rates of *QueryHits* to corresponding queries in both systems are compared. Firstly, ten different randomly generated queries belonging to favorites are submitted in both systems. And each query is executed ten times, and the average results are plotted in Figure 4(a).

As shown in Figure 4(a), the shapes of two curves are much more different, which means the peers that hold answers to queries are differently distributed in PeerIS and Gnutella. Furthermore, as a whole, the completion time of different queries in PeerIS is roughly the half of that in Gnutella. It results from two reasons. The first is that almost all answers to a query are very near to the requester in PeerIS, while, in Gnutella, the answers are randomly distributed. The second reason is that in PeerIS, all *QueryHits* are directly returned to the requester, while in Gnutella, they are returned to the requester along the same path as the query.

Secondly, the returning rates of QueryHits to each of corresponding query are also studied and the average rates are plotted in Figure 4(b). As the above figure shown, all the *QueryHits* in PeerIS are returned to the requester at higher rates than those of the responding *QueryHits* in Gnutella. Another subtle phenomenon is that most of the *QueryHits*



(a) The Completion Time of Different Queries.



(b) Completion Time of QueryHits.

Figure 4. Completion Time Experiments.

are very near the requester, and some of them share the same rate, which also reflects the distribution of a peer's *confidants* and *acquaintances*.

5.5 Distribution of Query

All of the queries in the previous experiments fall into one of the four categories of the favorite files. In this experiment, the effect of the distribution of queries is studied, i.e., some of the queries search for files of the four categories while the rest search for the randomly selected files. The experiment consists of seven groups of executions, and each of which are made up of ten different individual queries. In the first group, all the queries fall into the four categories of favorite files, while in the second group, nine queries are used to search the four categories of favorite files and the rest is used to search for non-favorites, and so on. Each query is run ten times and the average results are shown in Figure 5.

Figure 5 shows how the different query distributions affect the performance of different P2P systems. For PeerIS, with the percentage of queries of searching sharing favorite files decreasing, its performance deteriorates, while the performance of Gnutella almost keeps the same. Furthermore, when the percentage of queries searching "favorite" files decreases at about 50%, Gnutella outperforms PeerIs. It is because, in PeerIS, each peer just maintains those who

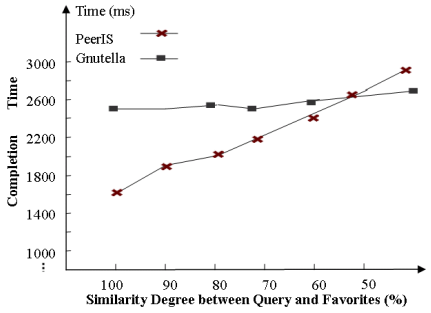


Figure 5. Compact of Query Distribution to Performance

share the similar favorites as its *confidants*, so that peers in PeerIS have less access to the "non-favorite" files; while in Gnutella, each peer just randomly selects its *confidants* and so do its *confidants*. Thus, its Performance is not sensitive to the distribution of the queries. However, for PeerIS, this situation can be alleviated by its evolution function.

6 Related Work

Three famous systems, i.e., Napster [7], Gnutella [4, 3] and Freenet [2] are regarded as the inspiration of recent P2P research and development. Pioneered by Napster, last two years has seen an explosive growth in the use of file-sharing software. This hybrid P2P system uses a centralized server to maintain meta-information. By using a server to maintain all the meta-information of files that are current available in the overall system, resource locating in Napster is efficient. However, since all locating operations need facilitating of its server, there may exist a single point of failure as well as scalability bottleneck. Gnutella is a fully distributed version of file sharing where there is no central entity. Both file-location and transfer are handled in a completely distributed manner. However, in order to locate a desired file, queries flood the network with exponential messages. Furthermore, all *QueryHits* and *failure Messages* are returned to the requester along the original path, which leads to low performance. Another obvious limitation of Gnutella is that the confidants of a given peer are randomly and statically defined, which hinders performance as well as the effectiveness of resource utility. In Freenet, another fully distributed file P2P system, a peer just forwards a query to one of confidants, and *QueryHits*, together with the requested file is returned to the requester along the same path as that of the query. This location strategy is obvious inefficient. Moreover, in the worst cases, the requested file cannot be located even if it actually exists. A common limitation of these three systems is that they just support coarse granularity of sharing with weak semantics. Fortunately, all the limitations of

these systems mentioned above are eliminated by our location mechanism.

In the location and routing research field of P2P computing, the most inspiring achievements should be Pastry [1], CAN [10], Chord [13], and Tapestry [16]. All of the systems support distributed hash functionality (DHT) [11]. In these systems, files are associated with a key (typically produced by hashing the file name) and each peer in the system is responsible for storing a certain range of the keys. Because of their efficiency, several projects are built upon them. However, this type of methods confronts many challenges. Besides thirteen open problems proposed by [11], we note there are still following challenges. Firstly, this strategy obviously ignores the autonomous nature of the peers in P2P systems, which of course is against the main idea of P2P. The second problem is that it just supports file level searching and sharing with weak semantics. Last but not the least, it also face the problems of hot spot and security.

[9] proposes the metric of relative importance of confidant, which is similar to our *Benefit-Cost of QueryHits*. However, our metric of relative importance is more comprehensive, which includes not only *Benefit-Cost of QueryHits*, but also similarity of behavior, relative connection status, and computing resources. Furthermore, our mechanism has solved the problem of *Group_ID*, which is the premise of location mechanism, while [9] does not make any contribution to it. In addition, [9] claims that its system is implemented upon Gnutella, while when confronting the isolation problem of a peer, it turns to a server, which is a serious contradiction. In PeerIS, there is never the problem, since each peer not only has a list of acquaintances, but also has LIGLO service. In addition, in [9], all *QueryHits* are returned to the requester along the query path, while in PeerIS, *QueryHits* are returned to the requester directly.

7. Conclusions

The grand vision of P2P computing is facing immense technical challenges, one of which is how to locate desired resources. However, recent achievements in this field have many common limitations, e.g., just supporting weakly semantic functions as well as ignoring autonomous nature of P2P nodes. In this paper, we define the similarity degree of favorites and behavior between peers, and relative importance; as well as present their computing formulae. Secondly, we propose a content-based location mechanism, which can not only maintain the autonomous nature of peers but also support approximate query and fine granularity of content sharing. By employing strategy of file classification, our mechanism can easily determine the similarity degree of favorites between peers as well as predict the category of future queries, and further guarantee the peers with high similar favorites clustered closely, so that efficient location can be achieved. Furthermore, a sophisticated tactic in this mechanism facilitates the system to evolve dynamically. Finally, we also have implemented our mechanism in the PeerIS and tested it. The experiment results further verify effectiveness of our mechanism At the same time, we plan to extend this mechanism in two directions. Firstly, we are about to extend our strategy to address the other two formats of information, i.e., video and audio. Secondly, in the depth direction, we will devise a suitable multicast routing strategy supported by our location mechanism.

References

[1] P. Druschel and A. Rowstron. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International conference on Distributed systems platforms (Middle ware)*, pages 329–350, 2001.

[2] Freenet Home Page. *http://freenet.sourceforge.com/*.

[3] FURI Home Page. *http://www.jps.net/williamw/furi*.

[4] Gnutella Development Home Page. *http://gnutella.wego.com/*.

[5] L. Gong. Peer-to-peer networks in action. In *IEEE Internet Computing Vol. 6*, pages 37–39, 2002.

[6] S. Gribble, A. Halevy, Z. Ives, M. Rodrig, and D. Suciu. What can databases do for peer-to-peer? In *WebDB Workshop*, 2001.

[7] Napster Home Page. *http://www.napster.com/*.

[8] W. S. Ng, B. C. Ooi, and K. L. Tan. Bestpeer: A self-configurable peer-to-peer system. In *International Conference on Data Engineering (ICDE) (Poster)*, 2002.

[9] M. K. Ramanathan, V. Kalogerakl, and J. Pruyne. Finding good peers in peer-to-peer networks. In *International Parallel and Distributed Computing Symposium*, 2002.

[10] S. Ratnasamy, R. Francis, M.Handley, R. Krap, J. Padye, and S.Shenker. A scalable content-addressable network. In *ACM SIGCOMM*, 2001.

[11] S. Ratnasamy, S. Shenker, and I. Stocia. Routing algorithm for dhts: Some opern questions. In *IPTPS*, 2002.

[12] G. Salton and M.E.Lesk. Computer evaluation of indexing and text processing. In *Journal of the ACM*, pages 8–36, 1968.

[13] I. Stocia, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM*, 2001.

[14] The BestPeer Project Home Page. *http://xena1.ddns.comp.nus.edu.sg/p2p/*.

[15] Y. Yang and X. Liu. A re-examination of text categorization methods. In *ACM SIGIR*, 1999.

[16] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-resilient wide-area location and routing. In *Technical Report UCB//CSD-01-1141, U. C. Berkeley*, 2001.