Journal of Network and Computer Applications 41 (2014) 441–450

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Review

Effectiveness of a replica mechanism to improve availability with Arrangement Graph-Based Overlay

Ssu-Hsuan Lu[a], Kuan-Ching Li[b], Kuan-Chou Lai[c], Yeh-Ching Chung[a]

[a] *Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan*
[b] *Department of Computer Science and Information Engineering, Providence University, Taichung, Taiwan*
[c] *Department of Computer Science, National Taichung University, Taichung, Taiwan*

ARTICLE INFO

Article history:
Received 2 April 2013
Received in revised form
2 November 2013
Accepted 23 November 2013
Available online 1 December 2013

Keywords:
Overlay network
Arrangement graph
Peer-to-peer
Replica
AGO

ABSTRACT

Peer-to-peer (P2P) overlay networks continue to evolve and grow to meet the challenges of a new age. Because peers can join or depart overlay networks at any time, researchers are particularly interested in how peers should be allowed to join overlay networks as well as how to minimize overhead in overlay networks. An Arrangement Graph-based Overlay (AGO) system can efficiently reduce system overhead by reducing the number of messages in a large-scale environment. However, AGO produces too many polling messages for the joining process. To address this issue, an enhanced joining strategy greatly improves the joining process of AGO to reduce the number of joining messages in this paper which is called the Enhanced Arrangement Graph-based Overlay (EAGO). Besides, for the purpose of the effectiveness of resource prefetching, a replica mechanism is employed to further improve routing performance. Experimental results indicate that EAGO reduces the number of joining messages by at least 20% compared with AGO, and it reduces the average number of routing hops due to the replica mechanism.

Contents

E-mail addresses: shlu@sslab.cs.nthu.edu.tw (S.-H. Lu), kuancli@pu.edu.tw (K.-C. Li), kclai@ntcu.edu.tw (K.-C. Lai), ychung@cs.nthu.edu.tw (Y.-C. Chung).

1084-8045/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.jnca.2013.11.005

1. Introduction

Although initially run on single servers, Internet services and computing functions are now divided across multiple servers to achieve the benefits of distributed architectures. For example, the Google search engine runs over hundreds of thousands of interconnected servers. A distributed architecture comprises several different technologies, including peer-to-peer (P2P) overlay networks. P2P systems and applications have attracted considerable attention in computer science research.

Peers in P2P overlay networks are not only clients but also servers; therefore, peers can distribute the overheads that occur in client-server architecture. Each peer can access data from other peers as well as share its own data. The convenience of P2P overlay networks significantly affects common tasks such as file sharing, VOIP, and streaming. The management of peer-to-peer connections is a fundamental issue that influences the efficiency of data transmission between peers. One important goal of P2P networks is to ensure that all clients provide resources (such as bandwidth, storage space, and computing power) so that when they join the system, the capacity of the entire system also increases.

The Arrangement Graph-based Overlay (AGO) (Lu et al., 2011) is a P2P overlay network that inherits the properties of the arrangement graph on which it is based (Chen et al., 2000; Chiang and Chen, 1998; Day and Tripathi, 1992, 1993a, 1993b; Hsieh et al., 1997). The arrangement graph is denoted by $A_{n,k}$, where k denotes the number of digits of a peer ID, and n denotes the range of each digit of the ID. Any two adjacent peers differ by only one digit, and each peer manages a neighbor table generated according to its peer ID. AGO consists of three main parts: *joining*, *departing*, and *routing*. In AGO, *bootstrap peer* plays an important role in the joining process which is composed of several peers. The bootstrap peer manages a *waiting peer pool* in order to maintain some number of existing peers in AGO.

Although AGO already greatly reduces the number of messages created in the joining process and reduces bandwidth consumption, new peers oftentimes still need to re-send several messages in the joining process. Therefore, this study seeks to enhance the AGO joining process and introduces a replica mechanism into AGO; the new proposed algorithm overlay is called the Enhanced Arrangement Graph-based Overlay (EAGO) (Lu et al., 2012).

EAGO retains the properties of AGO but significantly modifies the main part of the joining process while improving upon AGO's limitations. There are two major improvements of EAGO over AGO: (1) an enhanced joining strategy that reduces the abundance of polling messages associated with the joining process and (2) a replica mechanism that improves routing performance. In the enhanced joining strategy of EAGO, the waiting peer pool in the bootstrap peer is adjustable. Peers in this pool are removed when their neighbor tables are full, which makes it unnecessary for new peers to repeatedly attempt to rejoin. In this way, the large number of messages can be reduced by ensuring that neighbor tables of peers in the pool are not full.

Another contribution of EAGO is that it adds a replication mechanism to AGO. In general, replication facilitates increased data availability by minimizing overhead (Rzadca et al., 2010). EAGO utilizes the property of the vertex symmetry of the arrangement graph to achieve this goal. In EAGO, each peer assigns a file replica to the complement peer whose peer ID is $(n+1)$'s complement number where n is the parameter n of the arrangement graph. In the process of routing, the peer that attempts to discover a destination peer can also attempt to discover the complement peer of the destination peer. Besides, this study utilizes the concept of virtual peers to deal with vacant peers to make the arrangement graph can be kept full. According to the property of the P2P environment, the number of peers in the system cannot be always equal to $n!/(n-k)!$. The goal of this method is to let other existing peers act as agents of those vacant peers that do not exist in the arrangement graph yet. Those vacant peers who are managed by other physical peers are called *virtual peers*. After a new peer joins the system, it tries to discover information of its neighbors, and becomes agents of its vacant neighbors if it is needed. Basically, peers only need to be agents of their neighbors, so a physical peer may also acts as several virtual peers. By using this method, the arrangement graph can be kept full, and the routing hops of the arrangement graph can be kept within $\lfloor 3k/2 \rfloor$. In this way, routing performance and resource availability can be increased.

Some experiments were performed to demonstrate the proposed mechanism. The experimental results revealed that the enhanced joining strategy not only reduces the number of joining messages but also reduces bandwidth consumption under different churn rates (Churn rate, 2013). Although a few numbers of messages are created for maintaining virtual peers, the virtual peer mechanism can help the performance of the replica mechanism. The experimental results demonstrate that the replication mechanism used by EAGO reduces the number of routing hops and increases resource availability.

The remainder of this paper is organized as follows. Section 2 presents details on P2P overlay networks and AGO system. Section 3 describes the proposed EAGO, and some experimental results are shown in Section 4. Finally, conclusions and future work are discussed in Section 5.

2. Related work

P2P systems release the need for a central coordination point, and this makes systems more scalable and flexible. This section introduces P2P overlay networks in general and the AGO system in particular.

2.1. Peer-to-peer overlay network

P2P overlay networks have become a popular research topic in recent years. The P2P overlay network is an abstract virtual network based on a physical network. In this virtual network, any connection between two peers does not consider the actual connection topologies of the physical network but instead is initiated according to a specific algorithm. Additionally, P2P overlay networks must consider how to allow peers to join or depart from the network. There are many well-known systems that use P2P overlay networks (Lua et al., 2005); these systems can be divided into *structured* and *unstructured* networks according to the topology of the overlay networks.

In structured P2P overlay networks, overlay networks assign keys to data items and configure peers to comply with certain rules corresponding to these keys. The most common method involves the Distributed Hash Table (DHT) (Lua et al., 2005) to assign a unique pair set $\{key, value\}$ to peer positions and data, as demonstrated by Chord (Stoica et al., 2001, 2003), Pastry (Rowstron and Druschel, 2001), and Kademlia (Heep, 2010; Maymounkov and Mazieres, 2002). Unstructured P2P overlay networks (Haribabu et al., 2008) often use flooding (Jiang et al., 2003, 2008) and Time-To-Live (TTL) to make queries on overlay networks, such as Gnutella (Lua et al., 2005). This is because there is no any specific rule for distributing and querying resources. Therefore, some research has discussed how to route effectively in unstructured P2P overlay networks, such as Route Learning (Ciraci et al., 2009) and traceable gain matrix (TGM) (Xua et al., 2010). Furthermore, P2P overlay networks also can be applied to wireless ad hoc environment. In (Dhurandhera et al., 2011), authors tried to apply the bee algorithm to P2P file searching in mobile ad hoc networks.

Chord, a P2P overlay network, uses consistent hashing to assign keys to peers in order to establish a network. The consistent hashing function uses SHA-1 as the base function to assign an *m*-bit identifier to each peer and data key. The length of the identifier *m* must be sufficiently large to ensure that each peer has a unique key. As *m* is the number of the bits in the key/NodeID space, each peer maintains a routing table with up to *m* entries called the finger table. Identifiers are ordered on an identifier circle known as the Chord ring, and each peer in the Chord ring must maintain a finger table that records predecessors and successors.

Koorde, which is based on Chord, includes de Bruijn graphs (Kaashoek and Karger, 2003). Koorde also uses consistent hashing to map keys to peers and forward lookup requests on the identifier circle, which has an embedded de Bruijn graph. Each peer in the de Bruijn graph is represented by binary numbers of *b* bits. A peer and a key both have identifiers that are uniformly distributed in a 2^b identifier space.

Pastry uses a Plaxton-like prefix routing method to establish its overlay network. Each peer is assigned a *128*-bit peer identifier known as NodeID. NodeID is a circular space from 0 to $2^{128}-1$. When a peer joins the system, the peer is assigned a space within this circular space. Each peer in Pastry maintains a neighborhood set, a leaf set, and a routing table. The NodeIDs and IP addresses of $|M|$ peers that are closest in proximity to the local peer are included in the neighborhood set *M*. Each peer also has a leaf set *L*, which contains the *L*/2 numerically largest NodeIDs and the *L*/2 numerically smallest NodeIDs. A routing table records the NodeIDs that have the same prefix as its NodeID.

Kademlia uses a *160*-bit key space to assign each peer a NodeID, and {*key*, *value*} pairs are stored on peers with IDs close to that of the key. Kademlia uses a novel XOR metric to measure the distance between points in the key space. There are three parameters of Kademlia: *Alpha*, *B*, and *k*. *Alpha* is a small number representing the degree of parallelism in network calls; *B* is the size in bits of the keys used to identify nodes and store and retrieve data; and *k* is the maximum number of contacts stored in a bucket.

2.2. Arrangement graph

An arrangement graph is a generalized class of star graphs that preserves ideal qualities of the star graph topology. An arrangement graph can be represented as $A_{n,k}$, which is given by two parameters *n* and *k*, with $1 \le k \le n-1$. It is an undirected graph, and its peers are arrangements of *k* elements chosen from *n* elements. Furthermore, the edges of an arrangement graph between two peers correspond to arrangements differing by exactly one of their *k* elements. The following are some definitions and basic properties of arrangement graphs.

Definition 1. An (n, k)-arrangement graph $A_{n,k}$ is an undirected graph (V, E), where $V=\{p_1p_2\ldots p_k | p_i \in \langle n\rangle \text{ and } p_i \ne p_j \text{ for } i \ne j\} = P^n_k$, and $E=\{(p,q)|p,q\in V, \text{ and for some } i \text{ in } \langle k\rangle\ p_i \ne q_i \text{ and } p_j = q_j \text{ for } j \ne i\}$.

Basic properties. An (n, k)-arrangement graph is represented as $A_{n,k}$ where the number of peers is $n!/(n-k)!$, the degree is $k(n\text{-}k)$, and the diameter is $\lfloor 3k/2 \rfloor$.

$G=(V, E)$ comprises sets of peers, such as participating peers, and edges, such as overlay links. Each peer of $A_{n,k}$ is an arrangement with *k* digits out of *n* elements of $\langle n\rangle$. Note that edges connect peers that differ by exactly one of their *k* digits. An edge of $A_{n,k}$ that connects two arrangements that differ only in position *i* is called an *i*-edge, and $A_{n,k}$ is vertex- and edge-symmetric.

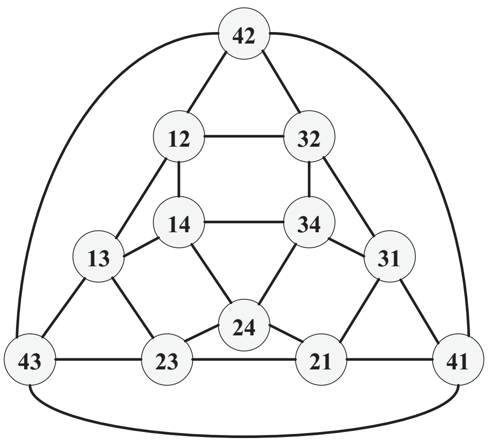

Fig. 1. (*4*, *2*)-Arrangement graph.

A (*4*, *2*)-arrangement graph is shown in Fig. 1. Peer 24 connects to Peers 21, 23, 14, and 34. Only one digit of Peer 24's ID differs from those of these four peers (i.e., 21, 23, 14, and 34). In other words, all peers with one different digit are connected, and each peer connects to four peers; this is calculated as $k(n\text{-}k)=4$.

2.3. The AGO system

AGO (Lu et al., 2011) provides a P2P overlay network environment that is developed from the arrangement graph. Therefore, AGO inherits some properties of the arrangement graph to deploy an overlay network and develop a routing algorithm. In this section, details on the construction of AGO and the routing algorithm are provided.

2.3.1. Overview of AGO

AGO is developed using the properties of the arrangement graph. AGO consists of three main parts: *joining*, *departing*, and *routing*. To join AGO, each peer needs to process a series of joining actions. Furthermore, the bootstrap peer serves as the AGO portal, which plays an important role in AGO. Each peer joins AGO by communicating with the bootstrap peer and the bootstrap peer has a pool called a waiting peer pool. This waiting peer pool is used to record a set of peers that already exist in AGO so that new peers can ask for the IDs of these existing peers.

When a new peer wants to join AGO, it must contact the bootstrap peer first and request the ID of an existing peer in the waiting peer pool. When the bootstrap peer receives the request from the new peer, it chooses a peer ID from the waiting peer pool and sends the peer ID back to the new peer. After the new peer receives the existing peer ID from the bootstrap peer, it sends a request to this existing peer and requests to be a neighbor. When the existing peer receives this request from the new peer, it checks whether its neighbor table is full. If the neighbor table is not full, it selects an unused peer ID and assigns this peer ID to the new peer. However, if its neighbor table is full, it selects one of its neighbors, relays this neighbor's information to the new peer, and instructs the new peer to request to be that neighbor's neighbor.

Once the new peer receives a reply from the existing peer, it will repeat the actions outlined above. If the new peer obtains a peer ID, the new peer becomes the neighbor of the existing peer and successfully joins AGO. However, if the new peer does not obtain a peer ID and receives an ID of another existing peer, then the new peer re-requests a peer ID from this peer to join AGO.

Whenever the new peer obtains a peer ID, the IDs of its neighbor table are generated according to its peer ID, and information on its neighbors becomes discovered by using a routing algorithm. Each peer also generates an ID list according to $A_{n,k}$ using the same specific algorithm, and each peer becomes aware of its related peers in the ID list. This ID list is used to discover resources when those peers do not exist.

Figure 2 illustrates the AGO joining process, which includes the following steps:

1. A new peer attempts to join AGO by sending a joining request to the bootstrap peer.
2. The bootstrap peer sends information of an existing peer in the waiting peer pool to the new peer.
3. After the new peer receives the information, the new peer sends a joining message to the existing peer.
4. The existing peer finds that its neighbor table is full, and it randomly selects an existing peer from its neighbor table and sends information on that existing peer to the new peer.
5. After the new peer receives information on this existing peer, the new peer re-sends a joining message to the existing peer.
6. The existing peer determines that its neighbor table is not full, selects an unused peer ID from its neighbor table, and sends the ID back to the new peer.
7. After the new peer receives the ID, the new peer uses it to generate its own neighbor table. The new peer places information on the existing peer into its neighbor table and uses a routing algorithm to discover information on other neighbors.

Conversely, when peers depart AGO, they must inform other peers about their departure. When a peer wants to leave the system, the departing peer informs the peer in the next position of the ID list and transmits any files that it manages to the next peer.

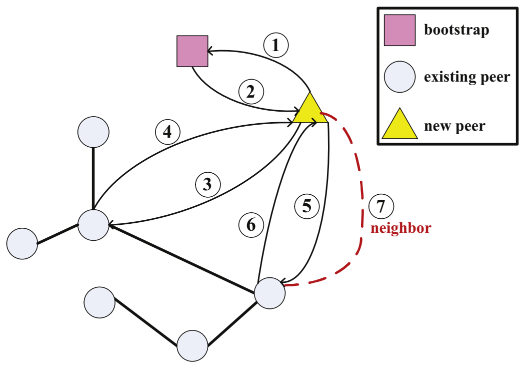

Fig. 2. Illustration of the joining processes in AGO.

```
Algorithm: Routing
Input: An ID that is a destination ID hashed from the filename
Output: The information of the peer whose ID is equal to the destination ID
Begin
When peer A receives an ID that is needed to lookup
if A's ID matches ID
    return its information;
else if any of A's existing neighbors' ID matches ID
    forward the request to that neighbor;
  else compares ID with A's existing neighbors digit by digit {
      if any of A's existing neighbors' ID is only one digit different from ID
          forward this request to neighbors;
      else {
          for( i=k-2 to floor(k/2) )
              if( there exist any A's existing neighbor whose ID has i digits the same as ID)
              {
                  forward the request to those neighbors;
                  break;
              }
      }
  }
End
```

Fig. 3. The routing algorithm pseudo-code.

The departing peer then sends requests to inform its neighbors and the bootstrap peer about its departure. When its neighbors receive the departure information, they remove the departing peer from their neighbor tables. The bootstrap peer also removes the departing peer from the waiting peer pool if necessary. To avoid unpredictable departures, each peer periodically sends requests to all of its neighbors. If the neighbors do not respond to the peer after some time, the peer assumes the neighbor that has departed and removes that peer from its neighbor table.

2.3.2. *Routing algorithm in AGO*

The routing algorithm is used by peers when joining overlay networks to determine whether the neighbors in their neighbor tables already exist. In addition, when a peer attempts to discover a resource, it learns the destination peer ID, which manages the resource by using DHT and tries to reach the destination peer by using the routing algorithm. AGO's routing algorithm was developed by utilizing the property that the IDs of any two adjacent peers differ by only one digit. Furthermore, the diameter of the arrangement graph is $\lfloor 3k/2 \rfloor$, so this property can be used to try to bind routing hops.

AGO's routing algorithm can be divided into two parts: *comparing* and *forwarding*. When a peer receives a request to discover the destination peer, it compares its peer ID with the destination peer ID. If they are the same, the peer relays its information to the peer who sent the request. However, if its peer ID is not the same as the destination peer ID, the peer begins to check the IDs of its neighbors. If any neighbor's ID is the same as the destination peer ID, the peer forwards the request to that neighbor.

If neither the ID of the peer nor the IDs of its neighbors are the same as the destination ID, the peer should forward this request to those neighbors that have a peer ID that is only one digit different from the destination peer ID. However, if no neighbor conforms to the above conditions, the peer may fail to meet the comparing digit requirement, meaning that the peer forwards the request to any neighbors with peer IDs that are two digits different from the destination ID. If there is still no neighbor who conforms to this stipulation, the peer forwards the request to any neighbors with peer IDs that are three digits different from the destination ID; this pattern continues until the criteria are satisfied. The worst-case scenario is to forward the request to neighbors with peer IDs that have at least $\lfloor k/2 \rfloor$ digits the same as the destination ID. In this way, AGO can avoid sending too many routing messages. Figure 3 shows the pseudo-code of the routing algorithm.

Figure 4 provides an example of the routing process with $A_{5,4}$ for the discovery of destination Peer 4251 by Peer 1453. When Peer 1453 tries to access data that Peer 4251 manages, Peer 1453 sends a lookup request to Peer 4251. Peer 1453 checks its neighbors according to the routing algorithm and finds that only Peer 1253 differs by two digits from Peer 4251. Therefore, Peer 1453 sends this request only to Peer 1253. Peer 1253 also follows the routing algorithm and sends the lookup request to Peer 1254. Finally, Peer 3251 finds that its neighbor Peer 4251 has the same ID as the destination ID, and the lookup request is sent to Peer

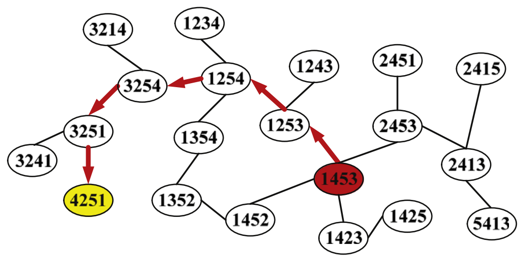

Fig. 4. An example of the AGO routing algorithm.

4251. According to the digit-filtering rule, Peer 4251 receives the lookup request and relays its information to Peer 1453. Additionally, each peer can also filter lookup requests to avoid dealing with the same lookup request multiple times.

3. Enhanced AGO system with an adjustable waiting peer pool

This paper enhances the waiting peer pool with respect to joining procedures and adds a replica mechanism to increase resource availability. Peers in the waiting peer pool are updated so that the joining process for new peers is more efficient and accomplished without the need to re-query. Replica files are stored by complement peers to improve searching performance. Furthermore, the virtual peer mechanism can avoid missing complement peers to put replica files and guarantee to bind routing hops. The following sections describe these two methods, which together enhance the performance of AGO.

3.1. Enhanced joining strategy

In the version of AGO described, peers in the waiting peer pool were fixed. Due to the size limitations of the neighbor tables of peers in the waiting peer pool, the new peer will need to query more frequently when the neighbor tables of peers in the waiting peer pool are full. Especially when there are large numbers of peers, individual peers must query several times and thus produce large numbers of messages. To eliminate this situation, an enhanced joining strategy is introduced. EAGO modifies the process of asking for peer IDs to improve the performance of the system as a whole.

EAGO makes the bootstrap peer's waiting peer pool adjustable such that peers in the waiting peer pool are not fixed. Peers in the waiting peer pool will be changed when their neighbor tables are full. When an existing peer in the waiting peer pool receives a joining request from a new peer, the existing peer checks whether its neighbor table is full. If its neighbor table is not full, the existing peer chooses an unused peer ID and assigns it to the new peer. The existing peer then checks its neighbor table again. If its neighbor table is full, the existing peer sends a message to the bootstrap peer to be removed from the waiting peer pool so that a new peer can be placed into the waiting peer pool.

The new peer sometimes sends a joining message to the existing peer before the bootstrap peer removes the existing peer from the waiting peer pool. In this case, the existing peer may find that its neighbor table is full when it receives the joining message from the new peer. To address this issue, the existing peer sends messages to the new peer and the bootstrap peer. The existing peer asks the new peer to send the joining request to the bootstrap peer again and asks the bootstrap peer to remove the existing peer from the waiting peer pool.

EAGO can therefore improve the AGO joining process by retaining only the neighbor tables of peers in the waiting peer pool that are not full. This method can greatly reduce the number of repetitive messages sent by new peers. The adjustable waiting peer pool enables the continuous changing of peers in the pool and ensures a more uniform peer distribution. Figure 5 provides an example of the EAGO joining process.

As shown in Fig. 5, the EAGO joining process consists of the following steps:

1–3. These three steps are the same as the first three steps of AGO.
4. This step differs from AGO. If the existing peer finds that its neighbor table is full while receiving a joining message from the new peer, the existing peer sends two messages: one to the new peer and one to the bootstrap peer.
   (a) One message informs the new peer to send the joining request to the bootstrap peer again.
   (b) The other message informs the bootstrap peer to remove this existing peer from the waiting peer pool.
5. The new peer sends the joining request to the bootstrap peer again.
6. The bootstrap peer sends the information of another existing peer to the new peer.
7. The new peer sends the joining message to another existing peer.
8. The existing peer finds that its neighbor table is not full, chooses an unused ID from its neighbor table, and assigns it to the new peer.
9. The new peer can then obtain an ID and discover information about its neighbors.

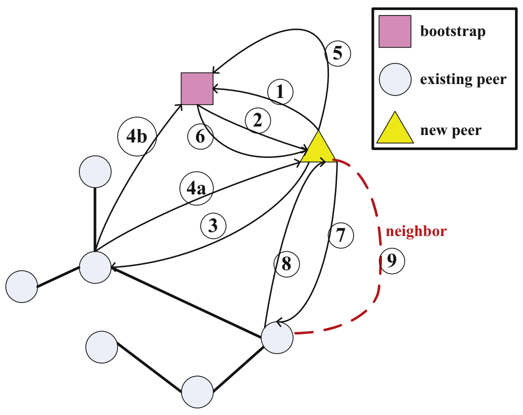

Fig. 5. The EAGO joining process.

3.2. The replica mechanism

The replica mechanism is one of the methods for overcoming the unpredictability of the behavior of peers. While each file is distributed to a certain peer, a replica is assigned to another peer. A good replica mechanism can provide the desired availability of data with a minimum number of replicas and can decrease the number of hops required to look up information. In EAGO, a replica mechanism is used to promote the routing algorithm performance. Because too many replicas will increase the difficulty of maintaining data correction, there is only a replica of each file is put on the complement peer. Each peer manages certain files and assigns replicas of files to its complement peer. The complement peer is the peer that has an ID that is the $(n+1)$'s complement number to its ID. For example, in the case of $A_{8,6}$, the complement peer of Peer 543612 is Peer 456387, so Peer 543612 assigns a file replica to Peer 456387. Therefore, Peer 456387 not only manages its files but also manages replica files of Peer 543612.

Peers in the arrangement graph are symmetric; therefore, if one peer is far from Peer A, the peer is near the symmetric peer of Peer A. This important property was used to design our replica mechanism. When a peer tries to access a certain file, it can access the file from the destination peer that manages the file as well as the file from the complement peer of the destination peer, the former of which manages the replica file. In this way, if the destination peer that manages the file departs or crashes, other peers can still access the replica file from the complement peer. Additionally, peers may be closer to the complement peer than to the destination peer that manages the file. Therefore, peers can route to the complement peer with fewer hops. By assigning replica files to complement peers, peers may be able to discover resources in a fewer number of hops.

Furthermore, each peer does not need to spend extra space to record where its own replica file is because each peer can directly calculate its complement peer ID according to its peer ID. When a peer tries to access a certain file, it can obtain the ID of the peer that owns that file and directly calculate the complement ID of the peer that owns the replica file. Therefore, the peer can send requests to both of the peers that own the original and replica files by using the routing algorithm. In this way, there is a greater chance of finding resources without increasing the overhead associated with recording the location of the replica file.

In order to avoid the situation that there does not exist the peer with ID being equal to $(n+1)$'s complement peer, the concept of virtual peers is used in our system. The vacant peers which are so called virtual peers are managed by other physical peers, those physical peers act as agents of those vacant peers. In a P2P environment, peers join and leave frequently, and virtual peers can make the arrangement graph of EAGO always full. Virtual peers in this system mean those peers that do not really exist and are managed by some other physical peers. Furthermore, the distance between any two peers is shorter than the value of diameter. Therefore, the longest distance between Peer *A* and Peer *A*'s complement peer is equal to the diameter. Now, there is a querying peer between these two peers, so this querying peer either is closer to Peer *A* or is closer to Peer *A*'s complement peer. The worst case is that the querying peer is in the middle of those two peers.

The following describe some actions for maintaining virtual peers. After a new peer joins the system and records its neighbors' information, it receives its information managed by a physical peer and the information of agents of those neighbor peers who do not exist. Besides, if the new peer finds that the agent of a neighbor peer is not neighboring relation, the new peer tries to obtain the right of managing that neighbor peer. The new peer also communicates with physical peers in its neighbor table, and distributes the loads of managing virtual peers. On the other hand, before a peer leaves the system, it chooses a physical neighbor peer to be the agent for it. The leaving peer also needs to choose some other physical peers to be agents for virtual peers it manages.

Figure 6 shows the same example as Fig. 4 but with the inclusion of the replica mechanism. As seen in Fig. 6, Peer 1453 needs to take five hops to reach Peer 4251 using the routing algorithm. Peer 2415 is the complement peer of Peer 4251, so a replica file of Peer 4251 is stored by Peer 2415. However, Peer 1453 needs to take only three hops to reach Peer 2415 using the routing algorithm. Therefore, this replica mechanism may allow peers to find files more quickly and with greater success.

3.3. Theorem analysis

In the arrangement graph, the maximum number of peers that AGO can accommodate is

$$N_{max} = \frac{n!}{(n-k)!}. \tag{1}$$

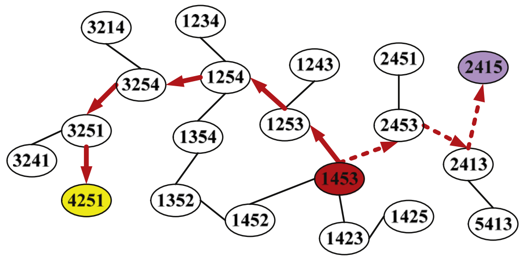

Fig. 6. An example of the EAGO routing algorithm with the replica mechanism.

Note that the diameter, which is defined as the longest distance between any two peers, is $\lfloor 3k/2 \rfloor$.

In general, the distance between any two peers is related to the number of different digits between those two peers. Assume that there are two peers and that Peer *A* tries to access data from Peer *B*. Peer *A*'s ID is $ID_A = a_1 a_2 a_3 \ldots a_k$, and Peer *B*'s ID is $ID_B = b_1 b_2 b_3 \ldots b_k$. If there are *x* different digits between Peer *A* and Peer *B*, then Peer *A* needs to make *x* hops to reach Peer *B*, where $x \le k$. This distance can be calculated as $A \oplus B$, and the aggregate of "1" indicates the distance.

For example, Peer *A*'s ID is $ID_A = 123456$, and Peer *B*'s ID is $ID_B = 127856$.

$$\rightarrow A \oplus B = 001100$$

There are two "1 s", which indicates that there are two different digits between Peer *A* and Peer *B*. Therefore, the hop count between Peer *A* and Peer *B* is equal to the number of "1 s" in $A \oplus B$.

Let us now consider the values of each digit of peer IDs. If there are *x* "1 s" in $A \oplus B$ but some values of these *x* digits are the same, then Peer *A* may need more than *x* hops to reach Peer *B*. For example, assume that Peer *A*'s ID is 123456 and that Peer *B*'s ID is 124356.

$$\rightarrow A \oplus B = 001100$$

However, because of the rule of IDs associated with the arrangement graph, Peer *A* cannot reach Peer *B* with only two hops; Peer *A* needs three hops to reach Peer *B*:

$$123456 \rightarrow 123756 \rightarrow 124756 \rightarrow 124356.$$

From the above discussion, it follows that

$$x \le Hop_{lookup}, \tag{2}$$

where Hop_{lookup} is the number of hops that Peer *A* needs to take to reach Peer *B*. According to the theorem associated with the arrangement graph, we know that

$$Hop_{lookup} \le \left\lfloor \frac{3k}{2} \right\rfloor. \tag{3}$$

From (2) and (3), we obtain

$$x \le Hop_{lookup} \le \left\lfloor \frac{3k}{2} \right\rfloor. \tag{4}$$

Next, let us consider the average number of hops that peers require to reach other peers. Assume that when Peer *P* tries to discover resources on Peer *Q*, there is a difference of *t* digits between these two peers. The number of peers that have *t* digits different from Peer *P* is

$$diff(t) = \binom{k}{t} \times \left[\prod_{n-k+1}^{n-k+t} + \sum_{i=1}^{t-1} \left((-1)^i \times \binom{t}{i} \times \prod_{n-k+1}^{n-k+t-i} \right) + (-1)^t \right]. \tag{5}$$

From (5), the expected value of the average number of hops is

$$E(x) = \frac{\sum_{t=1}^{k} (diff(t) \times t)}{N_{max}}. \tag{6}$$

We then have

$$E(x) \le E(Hop_{lookup}) \le \left\lfloor \frac{3k}{2} \right\rfloor. \tag{7}$$

Furthermore, each peer assigns a replica to its complement peer, which enables other peers to have a greater chance of finding resources with fewer hops. However, a peer may be far from the original peer that has the resource but close to the complement peer, which has the replica. For instance, in $A_{8,6}$, Peer *E* with $ID_E = 371542$ tries to find the resource on Peer *F* with $ID_F = 123456$. Peer *G* is the complement peer of Peer *F* with $ID_G = 876543$. The number of hops that Peer *E* needs to take to reach Peer *G* is less than that required to reach Peer *F*. Therefore, Peer *E* can obtain the

Table 1
Parameters of each overlay network.

Overlay	Parameters
AGO, EAGO	$n=8$, $k=6$
Chord	$m=14$
Koorde	$b=14$
Pastry	$b=4$, $l=16$
Kademlia	$Alpha=3$, $B=160$, $k=20$

resource from the complement Peer *G* instead of from Peer *F*, as demonstrated in (8).

$$Hop_{lookup} = \mathrm{Min}(Hop_{peerF}, Hop_{peerF'scomplementpeer}). \quad (8)$$

4. Experimental results

This section presents some experimental results on AGO, EAGO, and other P2P systems. OverSim (Baumgart et al., 2007, 2009; Munoz-Gea et al., 2009; The OverSim P2P Simulator, 2013) was used to evaluate the performance of these systems because it allows them to be evaluated based on the same environment setup.

4.1. Experimental environment

Both AGO and EAGO were implemented using OverSim. OverSim is an open-source simulation framework for building overlays and P2P networks on top of the OMNeT++ simulation environment (OMNeT, 2013), and it has a flexible underlying network scheme. Several structured and unstructured P2P systems and overlay protocols are contained in OverSim, such as Chord, Kademlia, Pastry, and Koorde. It is a powerful and widely used simulator for investigating P2P environments. Default values are used for all general settings provided by OverSim, such as network settings, to make all simulations run under consistent environment.

The following paragraph describes some of the parameters used in our experiments and provides descriptions of our simulations. AGO and EAGO were executed with $A_{8,6}$, and the parameters used in the other P2P overlay networks are shown in Table 1. These parameters are set to make the peer space of each overlay network is close to 20,000 peers. The overlay framework OverSim was employed to simulate all experiments with 1000–10,000 peers. Each test scenario was simulated 10 times, and average values were calculated by removing the maximum and minimum values. The fundamental nature of P2P overlay networks enables peers to join and depart frequently, so simulations were also executed with different churn rates. The churn rate is a measure of peers join and leave over a specific period of time (Churn rate, 2013), and all peers were assigned different lifetimes that varied between 1000 and 10,000 s according to the (Weibull distribution, 2013).

4.2. Number of messages created to build a system

When a peer joins a P2P overlay network, it needs to process a series of steps that include requesting a peer ID and discovering neighbors; these steps produce a large number of messages. Peers in both EAGO and AGO join systems by sending messages to peers in the waiting peer pool. In Chord, peers obtain peer IDs via hashing table and sending messages to neighbors. The joining processes of Koorde are similar to those in Chord but using different algorithm. In Pastry, it randomly assigns peer IDs according to the value of parameter *b* and generates related neighbor table. In Kademlia, peers also randomly obtain peer IDs and discover distances to other peers by computing as the exclusive or of the two peer IDs. Given that an excessive number of messages will affect the efficiency of these systems, the purpose of this experiment was to illustrate how many messages EAGO and other P2P overlay networks produce for peers joining systems.

The EAGO performance in building the system can be evaluated by comparing message amounts created by peers joining various systems. EAGO uses an enhanced joining strategy to improve the original processes of AGO; Fig. 7 illustrates the performance results of these improvements. Some P2P overlay networks created messages that were too large, such as Pastry records more information of other peers and needs to maintain that information. Therefore, Pastry produces a larger number of messages than other P2P overlay networks. The results are shown on a logarithmic scale to illustrate the corresponding relationship between these overlay networks. The details on the number of messages are presented in Table 2 for understanding real differences between these P2P overlay networks.

As seen in Fig. 7, the AGO system significantly reduces the number of joining messages created compared with other P2P overlay networks. Furthermore, EAGO yields significant reductions in the number of joining messages relative to AGO. Table 2 provides details on the number of messages created for each overlay network and shows that EAGO can reduce the number of messages by at least 20% compared with AGO. EAGO created less than 25% of the number of messages created in other overlay networks, and this reduction became more apparent for large peer groups. This result shows that EAGO can significantly reduce message amounts when building a system.

In the real world, peers frequently join and depart P2P overlay networks. Whenever a peer joins or departs a P2P overlay network, the peer produces several messages. When the number of peers becomes large, the number of these messages also becomes very large. These messages will consume the bandwidth of both peers and overlay networks, so some additional experiments were conducted to demonstrate how these messages are affected at different churn rates.

4.3. Churn rate

This subsection presents some experimental results to illustrate the effect of bandwidth consumption at different churn rates. The bandwidth consumption corresponds to the average number of sent and received message bytes per second for a given peer. These messages include joining messages, routing messages, and messages associated with the maintenance of neighbor tables. Because the fundamental nature of P2P overlay networks allows peers to join and depart at any time, peers must send many messages to maintain neighbor tables for joining or departing systems. Figure 8 shows the bandwidth consumption of each

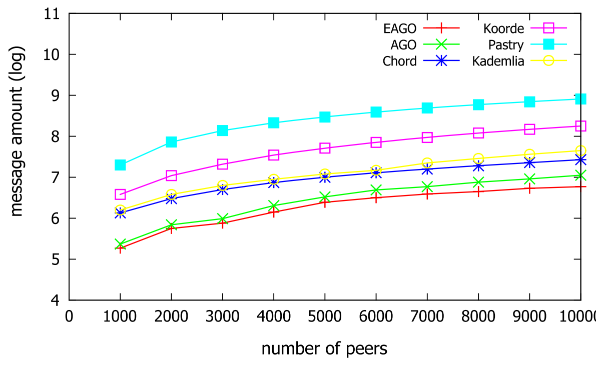

Fig. 7. Number of messages created by peers to join various systems.

Table 2
Detailed information on the number of messages created.

Peers	EAGO	AGO	Chord	Koorde	Pastry	Kademlia
1000	185628	232797	1359830	3838868	19863140	1567417
2000	558635	697350	3031425	10882911	72058328	3820027
3000	758737	982692	5045875	21086653	138043748	6365253
4000	1415022	2018987	7332288	34495810	213784198	8981605
5000	2459740	3292855	9908696	50985456	298445765	11904636
6000	3150803	4890596	12737604	70654803	389899887	14854760
7000	3896397	5837528	15866716	93662019	488145456	22600149
8000	4518217	7616313	19193316	119301299	591390248	28758395
9000	5362931	9165380	22878497	148048890	699317417	36224712
10,000	5930388	11347705	26818547	179722853	810098957	45037570

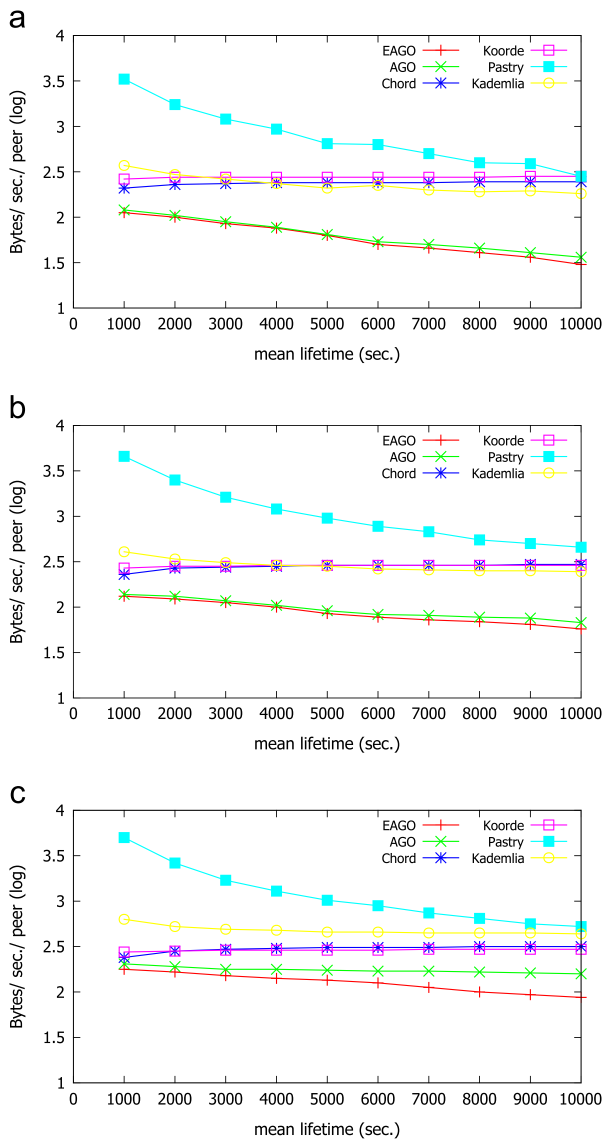

Fig. 8. Bandwidth consumption under different churn rates. (a) peer count=2000, (b) peer count=6000 and (c) peer count=10000.

overlay network at different churn rates for 2000, 6000, and 10,000 peers.

As shown in Fig. 8, lifetime varies between 1000 and 10,000 s, where a shorter lifetime means that peers join and depart systems more frequently. This results in the creation of a greater number of messages. The results from Fig. 8 are shown on a logarithmic scale to illustrate the corresponding relationship on bandwidth consumption between these overlay networks. As seen in Fig. 8, EAGO and AGO consume the least bandwidth, and EAGO consumes less bandwidth than AGO, especially when there are more peers in the system. Figure 8 shows that EAGO can reduce bandwidth consumption by approximately 10% relative to AGO and that EAGO consumes less than 40% of the bandwidth of other overlay networks. Therefore, EAGO can consume less bandwidth even when there are large numbers of peers.

4.4. *Average number of routing hops*

An efficient routing algorithm is very important in P2P overlay networks because peers often need to discover resources or look up the information of other peers. Routing also produces some messages, so our goal was to develop a routing algorithm without increasing system overhead in this study.

Figure 9 shows the average number of routing hops required to look up information for each P2P overlay network with different number of peers. In Fig. 9, the "AGO (Expected Value)" line corresponds to the expected average number of routing hops calculated using Eqs. (5) and (6) with $A_{8,6}$. The calculated expected average number of routing hops is equal to 5.25. To perform this simulation, the files are distributed to all peers to calculate the average number of routing hops.

Figure 9 shows that the average number of routing hops for AGO is approximately 5.5, which is slightly higher than the theoretical expected value derived from some of the special cases mentioned in the "Theorem analysis" section. However, the average number of routing hops for EAGO is approximately 4, which is less than the theoretical expected value. This result occurred because EAGO utilizes the replica mechanism, which allows peers to route with fewer hops. Although the average number of routing hops for EAGO is slightly higher than the average number of routing hops in Pastry, Pastry also creates many more messages than EAGO. Pastry records more information of peers, so this can benefit routing. The average number of routing hops for EAGO is constant, regardless of the number of peers in the system. This is because it is related to the value of parameter k according to the property of the arrangement graph. However, the average number of routing hops associated with other overlay networks increases slightly as the number of peers increases. Therefore, EAGO enables efficient routing without increasing system overhead, regardless of the number of peers.

4.5. *Request frequency*

In the previous subsection, the results of the average number of hops associated with different systems were presented. However, small numbers of routing hops sometimes result from the creation of a large number of messages, which produces large system overhead. In this subsection, the bandwidth consumption for different request frequencies is used to compare the performance of various systems.

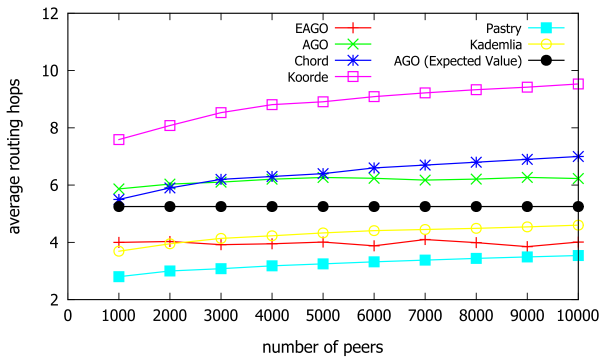

Fig. 9. Average number of routing hops required to look up information.

In P2P overlay networks, peers often send some requests to discover resources. If there are many peers and they send requests frequently, a large number of messages will be produced, thus consuming a large amount of bandwidth. To better illustrate this issue, the intervals between request messages were set to present the bandwidth consumption. Specifically, when the message interval time is short, peers send request messages more frequently. Figure 10 shows the bandwidth consumption of each P2P overlay network under different message intervals when the number of peers is 2000, 6000, and 10,000. These experiments were also performed with different churn rates. Because the trends of the different results are the same, only results with lifetimes of 5000 s are shown.

The results in Fig. 10 are shown on a logarithmic scale to illustrate the corresponding relationship between these overlay networks. As shown in Fig. 10, all of these overlay networks consume more bandwidth when the message interval time is short because more messages are created. EAGO and AGO consume the least bandwidth, and EAGO consumes less bandwidth than AGO. EAGO can greatly reduce the consumption of bandwidth, especially when there are more peers. Therefore, EAGO is more efficient in large-scale environments.

4.6. Discussion

The above simulations demonstrate that performance was greatly improved by enhancing the joining strategy and using a replica mechanism. The enhanced joining strategy can replace peers in the waiting peer pool. Compared with the joining strategy of AGO, this reduces the messages by at least 20% and consumes approximately 10% less bandwidth. Furthermore, the replica mechanism increases the availability of data and reduces the amount of hops needed for peers to route to destination peers by approximately 2 hops.

Besides, compared with other P2P overlay networks, EAGO significantly reduces the number of messages produced and reduces the number of average routing hops. From experimental results, EAGO and Kademlia almost have the same performance in routing hops, but the number of messages Kademlia produces is about 6 times of EAGO. Similarly, although the average routing hops of EAGO are a little more than those of Pastry, the number of messages Pastry created is at least 105 times of EAGO. Additionally, EAGO performs better than Chord and Koorde both in the number of messages created and average routing hops. In summary, EAGO can keep performance balance on the number of messages created and average routing hops. Furthermore, the experiments demonstrate that EAGO consumes less bandwidth under different churn scenarios and with different request frequencies, especially in large-scale environments. Therefore, EAGO promotes the performance in many aspects.

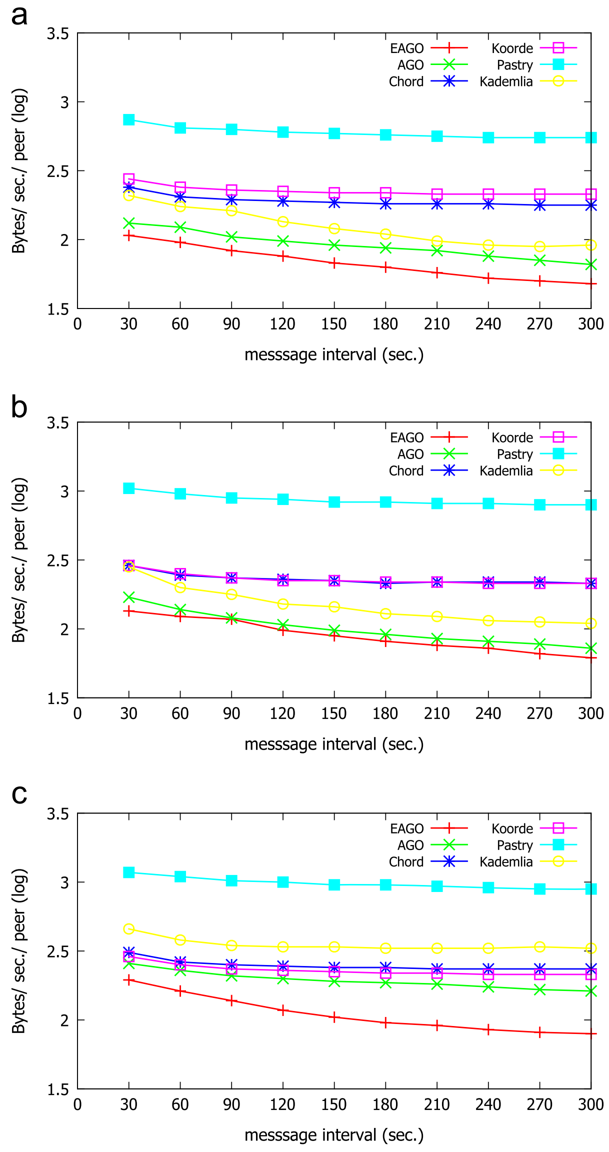

Fig. 10. Bandwidth consumption for different message intervals. (a) peer count = 2000, (b) peer count =6000, (c) peer count = 10000.

5. Conclusions and future work

This study proposes EAGO, which utilizes an enhanced joining strategy and integrates a replica mechanism to increase the resource availability of AGO. Furthermore, the virtual peer mechanism also is used to improve routing performance and avoid the missing problem of complement peers. In AGO, the joining process may force new peers re-send an excessive number of joining messages to existing peers. Although the average number of routing hops associated with AGO is close to the theoretical expected value, it is still slightly higher than the expected value. Therefore, an enhanced joining strategy and a replica mechanism were introduced to improve the joining process and increase resource availability.

The experimental results illustrate that the enhanced joining strategy significantly reduced the number of messages associated with attempts to re-join. This reduction can decrease system overhead for queries as well as bandwidth consumption. The enhanced joining strategy enables peers to join quickly without retrying several times and reduces the number of messages created. Furthermore, the strategy of adding replica to the complement peer using the property of the vertex symmetry of the arrangement graph reduces the average number of routing hops.

In the future, EAGO can be further improved. Just like other structured P2P overlay networks, the number of peers that overlay networks can contain is limited by certain parameters. Currently, the number of peers EAGO can contain is limited by (n, k). When the number of peers is above the capacity of EAGO, new peers must wait until some peers leave the system, which forces new peers to retry to join. To avoid this situation, EAGO will dynamically change values of (n, k) according to the number of peers in the system. EAGO is initialized with small values of (n, k), and the values are gradually increased. Based on this concept, this method can improve the efficiency of the system and reduce the number of routing hops. It is our hope that EAGO can be applied to other fields.

References

Baumgart I, Heep B, Krause S. Oversim: a flexible overlay network simulation framework. In: the proceedings of 10th IEEE global internet symposium (GI '07) in conjunction with IEEE INFOCOM. Anchorage, AK, USA; 2007. p. 79–84.

Baumgart I, Heep B, Krause S. Oversim: a scalable and flexible overlay framework for simulation and real network applications. In: the proceedings of IEEE ninth international conference on peer-to-peer computing (P2P'09). Seattle, WA, USA; 2009. p. 87–8.

Chen YS, Juang TY, Shen YY. Multi-node broadcasting in an arrangement graph using multiple spanning trees. In: the proceedings of the seventh international conference on parallel and distributed systems (ICPADS 2000). Japan: Iwate; 2000. p. 213–20.

Chiang WK, Chen RJ. On the arrangement graph. J Inf Process Lett 1998;66:215–9.

Churn rate. ⟨http://en.wikipedia.org/wiki/Churn_rate⟩[accessed 10/15/2013].

Ciraci S, Körpeoğlu İ, Ulusoy Ö. Reducing query overhead through route learning in unstructured peer-to-peer network. J Netw Comput Appl 2009;32(May): 550–67.

Day K, Tripathi A. Arrangement graphs: a class of generalized star graphs. Inf Process Lett 1992;42:235–41.

Day K, Tripathi A. Embedding grids, hypercubes, and trees in arrangement graphs. In: the proceedings of international conference on parallel proceeding (ICPP 1993). New York, USA; 1993. p. 65–72.

Day K, Tripathi A. Embedding of cycles in arrangement graphs. J IEEE Trans Comput 1993b;42(August):1002–6.

Dhurandhera SK, Misrab S, Pruthic P, Singhala S, Aggarwala S, Woungangd I. Using bee algorithm for peer-to-peer file searching in mobile ad hoc networks. J Netw Comput Appl 2011;34(Sep):1498–508.

Haribabu K, Reddy D, Hota C, Yla-Jaaski A, Tarkoma S. Adaptive lookup for unstructured peer-to-peer overlays. In: the proceedings of 3rd international conference on communication systems software and middleware and workshops 2008 (COMSWARE 2008). Bangalore, India; 2008. p. 776–82.

Heep B. R/Kademlia: recursive and topology-aware overlay routing. In: the proceedings of Australasian telecommunication networks and applications conference (ATNAC 2010). Auckland, New Zealand; 2010. p. 102–7.

Hsieh SY, Chen GH, Ho CW. Fault-tolerant ring embedding in faulty arrangement graphs. In: the proceedings of international conference on parallel and distributed systems (ICPADS 1997). Seoul, Korea; 1997. p. 744–49.

Jiang S, Guo L, Zhang X. Lightflood: an efficient flooding scheme for file search in unstructured peer-to-peer systems. In: the proceedings of the 2003 international conference on parallel processing (ICPP 2003). Kaohsiung, Taiwan; 2003. p. 627–35.

Jiang S, Guo L, Zhang X. Lightflood: minimizing redundant messages and maximizing scope of peer-to-peer search. J IEEE Trans Parallel Distrib Syst (TPDS) 2008;19(May):601–14.

Kaashoek MF, Karger DR. Koorde: a simple degree-optimal distributed hash table. In: the proceedings of the 2nd international workshop on peer-to-peer systems (IPTPS 2003), LNCS 2735. Berkeley, CA, USA; 2003. p. 98–107.

Lu SH, Lai KC, Li KC, Chung YC. Design and analysis of arrangement graph-based overlay systems for information sharing. In: the proceedings of the 3rd IEEE international workshop on management of emerging networks and services (IEEE MENS 2011) in conjunction with IEEE GLOBECOM 2011. Houston, TX, USA; 2011. p. 668–72.

Lu SH, Li KC, Lai KC, Chung YC. Arrangement graph-based overlay with replica mechanism for file sharing. In: the proceedings of the 12th international symposium on pervasive systems, algorithms and networks (ISPAN 2012). San Marcos, TX, USA; 2012. p. 192–200.

Lua EK, Crowcroft J, Pias M, Sharma R, Lim S. A survey and comparison of peer-to-peer overlay network schemes. J IEEE Commun Surv Tutor 2005;7(Second Quarter):72–93.

Maymounkov P, Mazieres D. Kademlia: a peer-to-peer information system based on the XOR metric. In: the proceedings of the first international workshop on peer-to-peer systems (IPTPS 2002). Cambridge, MA, USA; 2002. p. 53–65.

Munoz-Gea JP, Malgosa-Sanahuja J, Manzanares-Lopez P, Sanchez-Aarnoutse JC, Martinez-Rojo AM. Simulation of a p2p application using oversim. In: the proceedings of the first international conference on advances in future internet (AFIN 2009). Athens, Greece; 2009. p. 53–60.

OMNeT++ Network Simulation Framework. http://www.omnetpp.org/ [accessed 10/15/2013].

Rowstron A, Druschel P. Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems. In: the proceedings of the IFIP/ACM international conference on distributed systems platforms, Heidelberg (Middleware 2001). Heidelberg, Germany; 2001. p. 329–50.

Rzadca K, Datta A, Buchegger S. Replica placement in p2p storage: complexity and game theoretic analyses. In: the proceedings of IEEE 30th international conference on distributed computing systems (ICDCS 2010). Genoa, Italy; 2010. p. 599–609.

Stoica I, Morris R, Karger D, Kaashoek MF, Balakrishnan H. Chord: a scalable peer-to-peer lookup service for internet applications. In: the proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM 2001). San Diego, USA; 2001. p. 149–60.

Stoica I, Morris R, Liben-Nowell D, Karger D, Kaashoek MF, Dabek F, Balakrishnan H. Chord: a scalable peer-to-peer lookup service for internet applications. J IEEE/ACM Trans Netw 2003;11(February):17–32.

The OverSim P2P Simulator. ⟨http://www.oversim.org/⟩[accessed 10/15/2013].

Weibull distribution. ⟨http://en.wikipedia.org/wiki/Weibull_distribution⟩[accessed 10/15/2013].

Xua M, Zhoua S, Guanc J, Hud X. A path-traceable query routing mechanism for search in unstructured peer-to-peer networks. J Netw Comput Appl 2010;33 (March):115–27.