This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2327033, IEEE Transactions on Parallel and Distributed Systems

A Proximity-aware Interest-clustered P2P File Sharing System

Haiying Shen*, *Senior Member, IEEE,* Guoxin Liu, *Student Member, IEEE*, Lee Ward

Abstract—Efficient file query is important to the overall performance of peer-to-peer (P2P) file sharing systems. Clustering peers by their common interests can significantly enhance the efficiency of file query. Clustering peers by their physical proximity can also improve file query performance. However, few current works are able to cluster peers based on both peer interest and physical proximity. Although structured P2Ps provide higher file query efficiency than unstructured P2Ps, it is difficult to realize it due to their strictly defined topologies. In this work, we introduce a Proximity-Aware and Interest-clustered P2P file sharing System (PAIS) based on a structured P2P, which forms physically-close nodes into a cluster and further groups physically-close and common-interest nodes into a sub-cluster based on a hierarchical topology. PAIS uses an intelligent file replication algorithm to further enhance file query efficiency. It creates replicas of files that are frequently requested by a group of physically close nodes in their location. Moreover, PAIS enhances the intra-sub-cluster file searching through several approaches. First, it further classifies the interest of a sub-cluster to a number of sub-interests, and clusters common-sub-interest nodes into a group for file sharing. Second, PAIS builds an overlay for each group that connects lower capacity nodes to higher capacity nodes for distributed file querying while avoiding node overload. Third, to reduce file searching delay, PAIS uses proactive file information collection so that a file requester can know if its requested file is in its nearby nodes. Fourth, to reduce the overhead of the file information collection, PAIS uses bloom filter based file information collection and corresponding distributed file searching. Fifth, to improve the file sharing efficiency, PAIS ranks the bloom filter results in order. Sixth, considering that a recently visited file tends to be visited again, the bloom filter based approach is enhanced by only checking the newly added bloom filter information to reduce file searching delay. Trace-driven experimental results from the real-world PlanetLab testbed demonstrate that PAIS dramatically reduces overhead and enhances the efficiency of file sharing with and without churn. Further, the experimental results show the high effectiveness of the intra-sub-cluster file searching approaches in improving file searching efficiency.

Index Terms—P2P networks, File sharing system, Proximity awareness, File replication, Bloom filter.

1 INTRODUCTION

Over the past few years, the immense popularity of the Internet has produced a significant stimulus to P2P file sharing systems. For example, BitTorrent [1] constitutes roughly 35% of all traffic on the Internet. There are two classes of P2P systems: unstructured and structured. Unstructured P2P networks such as Gnutella [2] and Freenet [3] do not assign responsibility for data to specific nodes. Nodes join and leave the network according to some loose rules. Currently, unstructured P2P networks' file query method is based on either flooding [2] where the query is propagated to all the node's neighbors, or random-walkers where the query is forwarded to randomly chosen neighbors until the file is found. However, flooding and random walkers cannot guarantee data location. Structured P2P networks [4]–[7], i.e. Distributed Hash Tables (DHTs), can overcome the drawbacks with their features of higher efficiency, scalability, and deterministic data location. They have strictly controlled topologies, and their data placement and lookup algorithms are precisely defined based on a DHT data structure and consistent hashing function. The node responsible for a key can always be found even if the system is in a continuous state of change. Most of the DHTs require $O(\log n)$ hops per lookup request with $O(\log n)$ neighbors per node, where n is the number of nodes in the system.

- * Corresponding Author. Email: shenh@clemson.edu; Phone: (864) 656 5931; Fax: (864) 656 5910.

- *The first two authors are with the Department of Electrical and Computer Engineering, Clemson University, Clemson, SC, 29634. E-mail: {shenh, guoxinl}@clemson.edu. Lee Ward is with the Sandia National Laboratories. E-mail: lee@sandia.gov*

A key criterion to judge a P2P file sharing system is its file location efficiency. To improve this efficiency, numerous methods have been proposed. One method uses a super-peer topology [8]–[10], which consists of supernodes with fast connections and regular nodes with slower connections. A supernode connects with other supernodes and some regular nodes, and a regular node connects with a supernode. In this super-peer topology, the nodes at the center of the network are faster and therefore produce a more reliable and stable backbone. This allows more messages to be routed than a slower backbone and, therefore, allows greater scalability. Super-peer networks occupy the middle-ground between centralized and entirely symmetric P2P networks, and have the potential to combine the benefits of both centralized and distributed searches.

Another class of methods to improve file location efficiency is through a proximity-aware structure [9], [11], [12]. A logical proximity abstraction derived from a P2P system does not necessarily match the physical proximity information in reality. The shortest path according to the routing protocol (i.e. the least hop count routing) is not necessarily the shortest physical path. This mismatch becomes a big obstacle for the deployment and performance optimization of P2P file sharing systems. A P2P system should utilize proximity information to reduce file query overhead and improve its efficiency. In other words, allocating or replicating a file to a node that is physically closer to a requester can significantly help the requester to retrieve the file efficiently. Proximity-aware clustering can be used to group physically close peers to effectively improve efficiency. The third class of methods to improve file location efficiency is to cluster nodes with similar interests [13]–[19], which reduce the file location latency.

Although numerous proximity-based and interest-based super-peer topologies have been proposed with

1045-9219 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2327033, IEEE Transactions on Parallel and Distributed Systems

different features, few methods are able to cluster peers according to both proximity and interest. In addition, most of these methods are on unstructured P2P systems that have no strict policy for topology construction. They cannot be directly applied to general DHTs in spite of their higher file location efficiency.

This paper presents a Proximity-Aware and Interest-clustered P2P file sharing System (PAIS) on a structured P2P system. It forms physically-close nodes into a cluster and further groups physically-close and common-interest nodes into a sub-cluster. It also places files with the same interests together and make them accessible through the DHT `Lookup()` routing function. More importantly, it keeps all advantages of DHTs over unstructured P2Ps. Relying on DHT lookup policy rather than broadcasting, the PAIS construction consumes much less cost in mapping nodes to clusters and mapping clusters to interest sub-clusters. PAIS uses an intelligent file replication algorithm to further enhance file lookup efficiency. It creates replicas of files that are frequently requested by a group of physically close nodes in their location. Moreover, PAIS enhances the intra-sub-cluster file searching through several approaches. First, it further classifies the interest of a sub-cluster to a number of sub-interests, and clusters common-sub-interest nodes into a group for file sharing. Second, PAIS builds an overlay for each group that connects lower capacity nodes to higher capacity nodes for distributed file querying while avoiding node overload. Third, to reduce file searching delay, PAIS uses proactive file information collection so that a file requester can know if its requested file is in its nearby nodes. Fourth, to reduce the overhead of the file information collection, PAIS uses bloom filter based file information collection and corresponding distributed file searching. Fifth, to improve the file sharing efficiency, PAIS ranks the bloom filter results in order. Sixth, considering that a recently visited file tends to be visited again, the bloom filter based approach is enhanced by only checking the newly added bloom filter information to reduce file searching delay.

Note that although this work is for P2P file sharing systems, the techniques proposed in this paper can benefit many current applications such as content delivery networks, P2P video-on-demand systems, and data sharing in online social networks. Since the architecture of PAIS is based on a structured P2P system, its architecture cannot be used for unstructured P2P systems. However, PAIS's techniques for enhance efficiency of the intra-sub-cluster file searching can be used for unstructured P2P systems since nodes in a intra-sub-cluster are connected in an unstructured manner. The remainder of this paper is structured as follows. Section 2 presents a concise review of representative approaches for file location efficiency improvement in P2P systems. Section 3 describes PAIS, focusing on its structure construction and file searching algorithms. Section 4 describes the approaches that improve PAIS's intra-sub-cluster file searching. Section 5 presents trace-driven experimental results to show the effectiveness and efficiency of PAIS compared with other systems in both static and dynamic environments. Section 6 provides the conclusion for this paper.

2 Related Work

We discuss the related works most relevant to PAIS in three groups: super-peer topology, proximity-awareness, and interest-based file sharing.

Super-peer topology. FastTrack [10] and Morpheus [20] use super-peer topology. The super-peer network in [8] is for efficient and scalable file consistency maintenance in structured P2P systems. Our previous work built a super-peer network for load balancing [9]. Garbacki *et al.* [21] proposed a self-organizing super-peer network architecture that solves four issues in a fully decentralized manner: how client peers are related to super-peers, how super-peers locate files, how the load is balanced among the super-peers, and how the system deals with node failures. Mitra *et al.* [22] developed an analytical framework, which explains the emergence of super-peer networks on execution of the commercial P2P bootstrapping protocols by incoming nodes. Chordella [23] is a P2P system that is particularly designed for heterogeneous environments such as wireless networks. Sachez-Artigaz *et al.* [24] investigated the feasibility of super-peer ratio maintenance, in which each peer can decide to be a super-peer independently of each other. Liu *et al.* [25] proposed a hierarchical secure load balancing scheme in a P2P cloud system. It first balances the load among supernodes, and then depends on each supernode to balance the load among nodes under its management. Garbacki *et al.* [26] proposed a self-organizing supernode architecture to facilitate file querying. Each supernode caches the files recently requested by its children, and other peers send requests to the supernodes that can solve most of their requests.

Proximity-awareness. Techniques to exploit topology information in P2P overlay routing include geographic layout, proximity routing, and proximity-neighbor selection. *Geographic layout method* maps the overlay's logical ID space to the physical network so that neighboring nodes in the ID space are also close in the physical network. It is employed in topologically-aware CAN [11]. In the *proximity routing method*, the logical overlay is constructed without considering the underlying physical topology. In a routing, the node with the closest physical distance to the object key is chosen among the next hop candidates in the routing table. The entries of a routing table are selected based on a proximity metric among all the nodes that satisfy the constraint of the logical overlay (e.g., in Pastry [5], the constraint is the node ID prefix). This method has been adopted to Chord [4] and CAN [27]. *Proximity neighbor selection* selects the routing table entries pointing to the topologically nearest among all nodes with node ID in the desired portion of the ID space.

Genaud *et al.* [28] proposed a P2P-based middleware for locality-aware resource discovery. The works in [29] and [30] measured the inter-ISP traffic in BitTorrent and indicated the importance of locality-awareness traffic in reducing the traffic over long-distance connections. Guo *et al.* [31] examined traffic locality in PPLive and revealed that PPLive achieves high ISP level traffic locality. Shen and Hwang [32] proposed a locality-aware architecture with resource clustering and discovery algorithms for efficient and robust resource location in wide-area distributed grid systems. Lehrieder *et al.* [33] studied locality-awareness in scenarios with real-life, skewed peer distributions and heterogeneous access bandwidths of peers. Yang *et al.* [34] combined the structured and unstructured overlays with proximity-awareness for P2P networks to ensure the availability of searching results. Gross *et al.* [35] proposed a BitTorrent-like downloading scheme with locality-aware file searching and replication in order to supply a robust and fast downloading.

1045-9219 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2327033, IEEE Transactions on Parallel and Distributed Systems

Manzillo *et al.* [36] proposed the collaborative locality-aware overlay service, which reduces the transmit cost of ISPs by switching to the source inside the same ISP with the requester.

Interest-base file sharing. One category of interest-base file sharing networks is called schema based networks [16]–[18]. They use explicit schemas to describe peers' contents based on semantic description and allow the aggregation and integration of data from distributed data sources. Hang *et al.* [14] proposed a method for clustering peers that share similar properties together and a new intelligent query routing strategy. Crespo *et al.* [15] proposed a semantic overlay network (SON) based on the semantic relations among peers. Ruffo and Schifanella [37] studied the spontaneous communities of users in P2P file sharing applications and found that a family of structures show self-organized interest-based clusters. The works in [13], [38] consider node interest for publish and subscribe. Iamnitchi *et al.* [39] found the small world pattern in the interest-sharing community graphs, and suggested clustering common-interest nodes to improve file searching efficiency. Some works leverage the social network common-interest property for efficient file searching. Cheng *et al.* [40] proposed NetTube for P2P short video sharing. It clusters users with the same interests together for efficient peer assisted video delivering. Li *et al.* [41] proposed a P2P file sharing system based on social networks, in which common-multi-interest nodes are grouped into a cluster and are connected based on social relationship. Lin *et al.* [42] proposed a social based P2P assisted video sharing system through friends and acquaintances. Li *et al.* [43] grouped users by interests for efficient file querying and used the relevant judgment of a file to a query to facilitate subsequent same queries.

Liu *et al.* [44], [45] proposed online storage systems with peer assistance. The works in [46], [47] employ the Bloom filter technique for file searching. Despite the efforts devoted to efficient file location in P2P systems, there are few works that combine the super-peer topology with both interest and proximity based clustering methods. In addition, it is difficult to realize in DHTs due to their strictly defined topology and data allocation policy. This paper describes how PAIS tackles the challenge by taking advantage of the hierarchical structure of a DHT.

3 PAIS: A Proximity-aware Interest-clustered P2P File Sharing System

In our previous work [48], we studied a BitTorrent user activity trace [49] to analyze the user file sharing behaviors. We found that long distance file retrieval does exist. Thus, we can cluster physically close nodes into a cluster to enhance file sharing efficiency. Also, peers tend to visit files in a few interests. Thus, we can further cluster nodes that share an interest into a sub-cluster. Finally, popular files in each interest are shared among peers that are globally distributed. Thus, we can use file replication between locations for popular files, and use system-wide file searching for unpopular files. We introduce the detailed design of PAIS below. It is suitable for a file sharing system where files can be classified to a number of interests and each interest can be classified to a number of sub-interests.

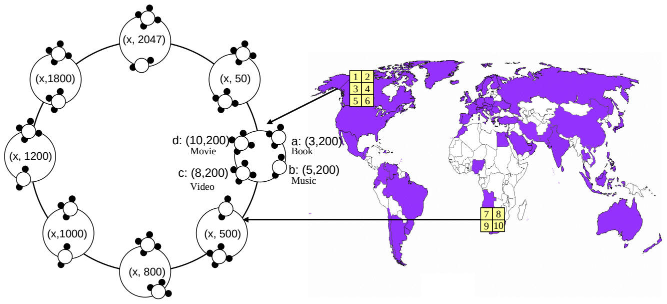


Fig. 1: The structure of PAIS.

3.1 PAIS Structure

PAIS is developed based on the Cycloid structured P2P network [7]. Cycloid is a lookup efficient, constant-degree overlay with $n{=}d{\cdot}2^d$ nodes, where d is its dimension. It achieves a time complexity of $O(d)$ per lookup request by using $O(1)$ neighbors per node. Each Cycloid node is represented by a pair of indices $(k, a_{d-1}a_{d-2}\ldots a_0)$ where k is a cyclic index and $a_{d-1}a_{d-2}......a_0$ is a cubical index. The cyclic index is an integer ranging from 0 to $d-1$, and the cubical index is a binary number between 0 and $2^d - 1$. The nodes with the same cubical index are ordered by their cyclic index mod d on a small cycle, which we call a *cluster*. All clusters are ordered by their cubical index mod 2^d on a large cycle. The Cycloid DHT assigns keys onto its ID space by a consistent hashing function. For a given key, the cyclic index of its mapped node is set to its hash value modulated by d and the cubical index is set to the hash value divided by d. A key will be assigned to a node whose ID is closest to the key's ID. Cycloid also has self-organization mechanisms to deal with node joins, departures, and failures. It has APIs, including `Insert(key,object)`, `Lookup(key)`, `Join()` and `Leave()`. Cycloid's routing algorithm involves three phases. A file request is routed along the cluster of the requester, between clusters, and along the cluster in the destination's cluster.

A node's interests are described by a set of attributes with a globally known string description such as "image" and "music". The strategies that allow the description of the content in a peer with metadata [16]–[18] can be used to derive the interests of each peer.

Taking advantage of the hierarchical structure of Cycloid, PAIS gathers physically close nodes in one cluster and further groups nodes in each cluster into sub-clusters based on their interests. We define a sub-cluster (SC) as a link structure within a network N given a set of links from client (c) to a particular supernode server (s). That is:

$$(SC_l = c_i, s_j \in N | \exists\ a\ link(c_i, s_j, l)),$$

Each SC_l supports functions: `join`(c_i, l) that links (c_i, s_j, l) are created between a server and a client and `leave`(c_i, l) where they are dropped.

The sub-cluster functions as a super-peer network that has one server and a number of clients connected to it. The servers are connected into a cluster in Cycloid. All nodes in a sub-cluster have the same Cycloid ID. Figure 1 illustrates the PAIS structure that shows clusters of nodes with common interests. Physically close nodes are in the same cluster, and common-interest nodes are grouped into one sub-cluster. The physically close nodes $1-6$ are mapped to cluster 200. The nodes interested in "book" are further grouped into sub-cluster a. All nodes in sub-cluster a have ID $(3, 200)$.

1045-9219 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2327033, IEEE Transactions on Parallel and Distributed Systems

3.2 PAIS Construction and Maintenance

Node proximity representation. A landmarking method can be used to represent node closeness on the network by indices used in [9]. Landmark clustering has been widely adopted to generate proximity information [11], [50]. It is based on the intuition that nodes close to each other are likely to have similar distances to a few selected landmark nodes. We assume there are m landmark nodes that are randomly scattered in the Internet. Each node measures its physical distances to the m landmarks and uses the vector of distances $< d_1, d_2, ..., d_m >$ as its coordinate in Cartesian space. Two physically close nodes will have similar vectors. We use space-filling curves [51], such as the Hilbert curve [50], to map the m-dimensional landmark vectors to real numbers, so the closeness relationship among the nodes is preserved. We call this number the *Hilbert number* of the node denoted by $\mathcal{H}$. The closeness of two nodes' $\mathcal{H}$s indicates their physical closeness on the Internet.

Node interest representation. Consistent hash functions such as SHA-1 is widely used in DHT networks for node or file ID due to its collision-resistant nature. When using such a hash function, it is computationally infeasible to find two different messages that produce the same message digest. The consistent hash function is effective to cluster messages based on message difference.

Clustering physically close and common-interest nodes. Based on the Cycloid topology and ID determination, PAIS intelligently uses cubical indices to distinguish nodes in different physical locations and uses cyclic indices to further classify physically close nodes based on their interests. Specifically, PAIS uses node i's Hilbert number, $\mathcal{H}_i$, as its cubical index, and the consistent hash value of node i's interest mod d ($S_i\%d$) as its cyclic index to generate node i's ID denoted by $(S_i\%d, \mathcal{H}_i)$. If a node has a number of interests, it generates a set of IDs with different cyclic indices. Using this ID determination method, the physically close nodes with the same $\mathcal{H}$ will be in a cluster, and nodes with similar $\mathcal{H}$ will be in close clusters in PAIS. Physically close nodes with the same interest have the same ID, and they further constitute a sub-cluster in a cluster.

Algorithm 1 Pseudo-code for node n joining in PAIS containing node $\dot{n}$.

```
n.join ($\dot{n}$){
1: generate IDs: $ID_1=(S_1, \mathcal{H}_n)$, ... , $ID_m=(S_m, \mathcal{H}_n)$
2: for $i = 0$ to $m$ do
3:    //find the server closest to the $ID_i$
4:    $s_i$=$\dot{n}$.lookup_server($ID_i$);
5:    if $n$ is a regular node then
6:       //take $s_i$ as its server
7:       $server_{ID_i}$=$s_i$;
8:       $server_{ID_i}$.join($n$, $l$);
9:    else
10:      //n is a supernode
11:      if $s_i$ is a supernode then
12:         $s_i$.addto_backuplist($n$);
13:      else
14:         //replace $s_i$ because it is a temporary supdernode
15:         $n.clientlist_{ID_i}$=$s_i.clientlist_{ID_i}$;
16:         $n.backpulist_{ID_i}$=$s_i.backuplist_{ID_i}$;
17:         $s_i$.remove_clientlist($ID_i$);
18:         $n$.join($s_i$, $l$);
19:         initialize routing table;
20:      end if
21:   end if
22: end for
23: }
```

Algorithm 2 Pseudo-code for node n leaving PAIS.

```
n.leave {
1: //assume node $n$ has $m$ interests
2: for $i = 0$ to $m$ do
3:    if it is the server in the sub-cluster of interest $i$ then
4:       if it has a supernode(s) in its backup list then
5:          find supernode from its backuplist to replace itself
6:          notify its clients about the server change
7:       else
8:          notify its clients to rejoin in the system
9:       end if
10:      execute leaving function in the Cycloid DHT
11:   else
12:      notify its server about its departure
13:   end if
14: end for
15: }
```

PAIS construction and maintenance. When node i joins the system, if there already exist nodes with IDs equal to $(S_i\%d, \mathcal{H}_i)$, and node i is a regular node, it becomes a client in the sub-cluster. If node i is a supernode, it becomes a backup for the server in the sub-cluster. Before the server leaves, one of the backups replaces the leaving server. If there is no node with IDs equal to $(S_i\%d, \mathcal{H}_i)$ and node i is a supernode, it becomes the server of the sub-cluster, and other newly-joined nodes with IDs $(S_i\%d, \mathcal{H}_i)$ will connect to it. If node i is not a supernode, it temporarily functions as the server until there is a joining supernode to replace it.

The clusters in PAIS function as a super-peer network. The server in a sub-cluster acts as a centralized server to a subset of clients by maintaining an index of files in the clients. Clients submit queries to their server and receive file location results from it like a hybrid system. Servers are also connected to each other as peers in a Cycloid. Servers route messages over this overlay and submit and answer queries on behalf of their clients and themselves.

To build each peer's routing table in the Cycloid, PAIS uses proximity-neighbor selection method. That is, it selects the routing table entries pointing to the nearest physical nodes among all nodes with IDs in the desired portion of the ID space. As a result, in PAIS, the logical proximity between neighbor abstractions derived from the overlay approximately matches the physical proximity information. Due to the uneven distribution of nodes in physical space, nodes may not be distributed with balance in the ID space of PAIS. The imbalance of node distribution will not generate adverse effects on file location efficiency in Cycloid [7]. Hence, it will not adversely affect the file location efficiency in PAIS. Algorithms 1 and 2 show the pseudocode for node join and departure in PAIS, respectively.

As normal structured P2P systems, PAIS uses stabilization to deal with node dynamism. Specifically, each server probes its routing table entries and predecessor periodically to make sure they are correct. If one of its neighbors fails to respond during a certain time period T, the server finds and connects to a new neighbor. In a sub-cluster, a server selects a secondary server from its backups that will replace it upon its departure or failure. It also notifies all clients about the secondary server. Before a server leaves, it requests the secondary server to be the new server and notifies all clients. The clients then connect to the new server. To handle the influence of a server failure on its clients, PAIS uses lazy-update. Specifically, each client probes its server periodically. If a client c does not receive a reply from its

1045-9219 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2327033, IEEE Transactions on Parallel and Distributed Systems

server s during T, c assumes that s fails, and connects to the secondary server. To further improve the reliability, multiple secondary servers can be used.

Performance analysis. Unlike other methods that use broadcasting to cluster nodes, PAIS leverages the DHT ID determination to cluster nodes based on their proximity and interest, thus reducing the overhead for structure construction. With the assumption that there are n_s servers in PAIS, we analyze the overhead of node dynamism in PAIS and achieve the following results.

Proposition 3.1: In PAIS, with high probability[1], a node join will incur overhead of $O(\log^2 n_s)$ messages.

Proof: When a node joins in PAIS, to find its closest server for one of its interests, $O(\log n_s)$ messages are required. If the new node joins as a server, another $O(\log^2 n_s)$ messages are required for neighbor update. Thus, if the node has m interests, the number of messages needed is $m(O(\log n_s) + \alpha O(\log^2 n_s)) \approx O(\log^2 n_s)$ in which α is the probability that the new node joins as a server. □

Proposition 3.2: In PAIS, w.h.p., a node departure will incur an overhead of $O(\log^2 n_s)$ messages, and a node failure will incur an overhead of $O(\log N_s)$ messages.

Proof: According to the PAIS node leaving algorithm, a leaving client only needs $O(1)$ message (i.e. notifying its server). If a leaving server has a backup supernode, it needs $O(\log^2 n_s)$ messages; otherwise, its clients need to rejoin the system again. Each client joining requires $O(\log n_s)$ messages. Therefore, the average number of messages caused by a node leaving is $O(1) \times \beta + (1-\beta)((1-\gamma)O(\log^2 n_s) + \gamma \times O(\log N_s)) \approx O(\log^2 n_s)$ where β is the percent of clients among all nodes, and γ is the probability that a sever has no backup supernode. It is easy to derive that a node failure incurs an overhead of $O(\log n_s)$ messages. □

3.3 File Distribution

As physically close and common-interest nodes form a sub-cluster, they can share files between each other so that a node can retrieve its requested file in its interest from a physically close node. For this purpose, the sub-cluster server maintains the index of all files in its sub-cluster for file sharing among nodes in its sub-cluster. A node's requested file may not exist in its sub-cluster. To help nodes find files not existing in their sub-clusters, as in traditional DHT networks, PAIS re-distributes all files among nodes in the network for efficient global search.

In PAIS, file ID is determined using the same way in Cycloid. That is, a file's cyclic index is its key's hash value modulated by d and its cubical index is set to the hash value divided by d, represented as $(H\%d, H/d)$, where H is consistent hash value of its key. A file's key can be its name or a string describing its contents. The file key must be consistent with the node interest. A node stores its files to the system via Cycloid interface `Insert(fileID,file)`. According to Cycloid key assignment policy, each sub-cluster is responsible for the files whose cyclic indices fall into the key space sector it supervises. Thus, files with similar keys will be in the same sub-cluster in a cluster. The supernode in a sub-cluster further distributes files among its clients in balance. For example, in Figure 1, a file with key "book" has ID $(3, 200)$, then it will be stored in a node in sub-cluster a. In node joins and departures, the files are transferred between nodes based on the key assignment policy.

1. An event happens with high probability (w.h.p.) when it occurs with probability $1 - O(n_s^{-1})$.

When a node joins in the system, it needs to distribute its files according to the file distribution protocol. These files can be accessed by the Cycloid routing algorithm through `Lookup(fileID)`. A newly joined node also needs to report its files corresponding to each of its interests to the server of each interest sub-cluster in order to enable other common-interest nodes to easily retrieve their requested files from physically close nodes. If a node leaves gracefully, it notifies other nodes to delete its files by `Lookup(fileID)`, and notifies its servers to delete the index of its files. If a server notices that a client failed, it removes the file index of the failed client.

In PAIS, if node i becomes very interested in file f in its cluster, it joins the sub-cluster $(H_f\%d, \mathcal{H}_i)$. If node i is not interested in a file category any longer, it departs the sub-cluster of the interest. By this way, PAIS tunes to time-varying node interest and file popularity dynamically.

File replication. PAIS relies on file replication to further improve its file location efficiency. Basically, when the request frequency of a file from a cluster of nodes with $\mathcal{H}$ exceeds a predefined threshold t, the file owner replicates a file in a node closest to $(H\%d, \mathcal{H})$ in that cluster. Creating a replica for a group of nodes with high accumulated visit rate on the file, so the file querying of the nodes can be significantly expedited; meanwhile, the nodes can take advantage of the file replicas.

Algorithm 3 Pseudo-code for looking up file f in PAIS.

```
n.lookup (key){
1:  get file f's key;
2:  get file f's ID $(H\%d, H/d)$;
3:  if $key \in interests$ then
4:      send request to its server of sub-cluster $H\%d$;
5:      if receive positive response from the server then
6:          exit;
7:      end if
8:      $Lookup(H\%d, \mathcal{H}_n)$;
9:      if receive negative response then
10:         //use Cycloid routing algorithm
11:         $Lookup(H\%d, H/d)$;
12:     end if
13: end if
14: }
```

3.4 File Querying Algorithm

The file querying algorithm has several stages: intra-cluster searching (consisting of intra-sub-cluster searching and inter-sub-cluster searching) and inter-cluster searching (i.e., DHT routing). If the intra-sub-cluster searching fails, PAIS relies on inter-sub-cluster searching. If the inter-sub-cluster searching fails, it will depend on DHT routing for file searching.

When node i wants to retrieve a file, if the file's key is one of the requester's interest attributes, it uses the intra-sub-cluster searching. Node i sends the request to its server in the sub-cluster of the interest. Recall that the server maintains the index of all files in its sub-cluster. Every time a server receives a request, it checks if its sub-cluster has the requested file. If yes, the server sends the file location to the requester directly. If the file's key is not one of the requester's interest attributes, node i checks the existence of the file or a replica of the file in its cluster (i.e., inter-sub-cluster searching). If there is a replica of the file, It should be stored in a sub-cluster closest to ID $(H\%d, \mathcal{H}_i)$. Therefore, the

1045-9219 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2327033, IEEE Transactions on Parallel and Distributed Systems

requester sends a request with $(H\%d, \mathcal{H}_i)$ as a target. The request is forwarded along the servers in each sub-cluster in the requester's cluster. If there is no requested file or replica of the requested file, the file request routing is performed based on Cycloid routing algorithm (i.e., inter-cluster searching). Then, node i calculates the ID of the file $(H\%d, H/d)$ and sends out a message of `Lookup(fileID)`. This routing algorithm does not lead to more overhead. Routing among physically close nodes greatly improves file location efficiency.

For example, in Figure 1, different files are classified into different sub-clusters based on their keys. When a node queries for a file in "book", it sends request `Lookup(3,200)` to its server. The requester receives the location of the file from the server if there is a requested file in its sub-cluster. Otherwise, the request is routed along its own cluster. Each server in the sub-cluster of the cluster checks if there is a requested file. If there is no requested file in the cluster, the request will be routed based on Cycloid routing algorithm which will forward the request to sub-cluster a. Then, the server of the sub-cluster a replies to the requester of the location of the file. Algorithm 3 shows the pseudocode of file lookup in PAIS. Proposition 3.3 demonstrates the efficiency of the file location in PAIS.

Proposition 3.3: W.h.p., the lookup path length for a file query in PAIS is bounded by $O(d)$ hops.

Proof: In PAIS, it takes a requester one hop to inquire its server for a requested file, and takes $O(d)$ to check a possible replicated file. Using the Cycloid routing algorithm, the path length is $O(d)$; therefore, w.h.p., the lookup path length for a query is bounded by $O(d)$ hops. □

4 Intra-sub-cluster Querying Enhancement

4.1 Sub-interest based File Querying

The cyclic index k indicates the interest of the nodes inside a sub-cluster. Since k is bounded by d, d must be no smaller than the number of interests in the system. With fine-grained interest classification, d must be a very large number. For example, d must be no smaller than 366 for the BitTorrent application with 366 fine-grained file interests [49]. However, a large d leads to increased lookup path length and hence lower efficiency in file querying based on Proposition 3.3. If PAIS has a small d, it then has coarse-grained interest classification, which leads to fewer sub-clusters in a cluster and much more peers inside a sub-cluster. This will introduce a long queuing delay and even single point of failure due to supernode congestions, since nodes depend on the supernode for intra-cluster searching. Thus, a small d decreases the intra-cluster searching efficiency. As a result, d represents a trade-off between the inter-cluster and intra-cluster searching efficiency.

To achieve high efficiency in both inter-cluster and intra-cluster searching, PAIS chooses a relative small d with coarse-grained interest classification and uses an additional method to improve the intra-cluster searching. A small d improves the inter-cluster file searching efficiency. Because a coarse-grained interest can be further be classified to a number of fine-grained interests, PAIS further clusters nodes inside a sub-cluster into sub-interest groups. For example, interest "'Music" can be further classified into sub-interests "Classic", "Jazz", "Pop" and so on. Using the supernode selection

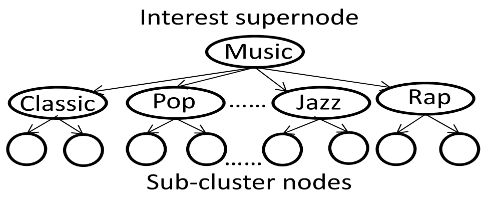


Fig. 2: Sub-interest based node clustering in a sub-cluster.

methods, a supernode is selected to be responsible for responding file queries inside each sub-interest group. The sub-interest supernode maintains an index of files residing in the nodes in the sub-interest group for file querying. The sub-cluster supernode records each sub-interest and its responsible supernode. As shown in Figure 2, inside a sub-cluster, nodes form a two layer tree. Nodes in a sub-interest group connect to the sub-interest supernode, and sub-interest supdernodes connect to the sub-cluster supernode. When a node in a sub-interest group A queries for a file in another sub-interest group B, group A's supernode will forward the request to the sub-cluster supernode (i.e., tree root), which then will forward the request to group B's sub-interest supernode. Then, group B's sub-interest supernode forwards the request to the node in its group that has the requested file if it exists. By splitting the workload on a single supernode among several supernodes of sub-interests, the queuing delay and the congestion on the sub-cluster supernode can be reduced. As a result, PAIS improves the intra-cluster searching efficiency as well as inter-cluster searching efficiency.

4.2 Distributed Intra-sub-cluster File Querying

There may be still tens of thousands of nodes inside a sub-interest group that have the same sub-interest and are in the same location [48]. Then, the file querying may become inefficient due to the sub-interest supernode overload or failure. Thus, though the sub-interest based file querying improves querying efficiency, it is still not sufficiently scalable when there are a very large number of nodes in a sub-interest group. We then propose a distributed intra-sub-cluster file querying method in order to further improve the file querying efficiency. In this method, nodes inside a large sub-interest group first search files among their neighbors in a distributed manner; if the search fails, the file requester then forwards its request to the sub-interest supernode as a complementary method. Below, we present how to connect the nodes in a sub-interest group for distributed file querying.

Nodes in a P2P system have heterogeneous bandwidth capacities. If a node receives many file requests but it has very limited bandwidth capacity, there will be a delay in the request responses. Requesting a file from a node with high bandwidth capacity can reduce file querying delay. Thus, PAIS classifies nodes in a sub-interest group to different classes based on the bandwidth capacity; nodes in the same class have similar bandwidth capacity. Let us use $\{C_1, C_2 ...\}$ to represent the classes in descending order of bandwidth capacity; nodes in class C_m have higher capacity than those in class C_{m+1}. Then, a node in class C_{m+1} connects to nodes in class C_m, that is, lower capacity nodes connect to higher capacity nodes.

PAIS needs to handle the node dynamism in building an overlay for a sub-interest group. When a node belonging to class C_{m+1} joins in a sub-interest group, the sub-interest supernode randomly selects M nodes in class C_m. The random selection is to balance the searching

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2327033, IEEE Transactions on Parallel and Distributed Systems

load from children among parent nodes. When a node in C_1 joins in the system, the supernode randomly selects M other nodes in C_1 to be its neighbors. When a non-leaf node leaves, it needs to notify its neighbors and the sub-interest supernode. The sub-interest supernode will assign the children new neighbors.

In the file searching, nodes send their requests to their parents with TTL (or neighbors in C_1), which further forward the request to their parents (or neighbors in C_1). Nodes asking their parents for files means higher capacity nodes providing files to lower capacity nodes. If a node has a higher capacity, it has more directly and indirectly connected children in the overlay; that is, it is likely to receive more requests. As a result, the file responding load is distributed among nodes based on their capacity; higher capacity nodes received more requests. If a node cannot find its requested file, it then broadcasts its requests to its children, which further forward the requests to their children. This process repeats until the file is located or the search fails. This step helps find a file, which is not in a higher capacity node but in a lower capacity node. If the search fails, the node resorts to the sub-interest supernode for the file.

4.3 Proactive File Information Collection based File Querying

In the distributed file searching in a sub-interest overlay, if a node proactively collects the information of files in its neighborhood, it can find file holders quickly from the collected information if they exist in the neighborhood. Then, the file search delay can be reduced and file search success rate can be enhanced. A simple method for a node to represent its file information is to use the names of all files held by it. In the file information collection process, each node sends its file information to its neighbors. To limit the file information collection overhead, each node's file information is forwarded for only TTL hops. It means that a node can know the file information of nodes within TTL hops. Thus, after a node collects all file information from its neighbors within TTL hops, it can check whether its requested files exists within its TTL hops and which nodes have this file. Then, in the file querying, after a node initiates a request, it checks its file information to find file holders; if it does not have the information of the queried file, it resorts to the supernode. As the file information directly contains file names and file holders, we refer to this file querying method as *FileName*.

4.4 Bloom Filter based File Querying

In the *FileName* method, the information exchange between neighbors may introduce a high network overhead. Also, the total size of the exchanged messages increases exponentially as TTL increases. To constrain the overhead in the file information collection, we further propose a method called *BloomFilter* that uses the counting bloom filter technique [52] to compress the exchanged messages. As shown in Figure 3, a counting bloom filter is represented by an array with each array position denoting an index. The bloom filter method has a number of hash functions. To feed a file to a bloom filter, the file's key is hashed by these hash functions. Then, the courter in the array position corresponding to each hash value is increased by one. The value in the k^{th} array position of the bloom filter represents the number of files that have $H_j(F_i) = k$ $(j = 1, 2, ...)$, where

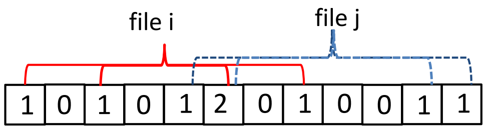


Fig. 3: The counting bloom filter.

$H_j()$ represents the j^{th} hash function and F_i represents a file fed into the bloom filter. For example, the counting bloom filter in Figure 3 has four hash functions. After hashing by the four hash functions, file i gains hash values 1, 3, 6 and 8, and file j gains hash values 5, 6, 11 and 12. Thus, the counters in the corresponding array positions are incremented by one. A newly added or deleted file needs to be added into or deleted from the bloom filter with its key information. We use $\mathbf{v}$ to represent the bloom filter array (e.g., Figure 3). We use $c_{H_j(F_i)}$ to denote the counter value for hash value $H_j(F_i)$ in $\mathbf{v}$. To decide whether a file F_i is in a bloom filter, the hash values $H_j(F_i)$ $(j = 1, 2, ..., K)$ are calculated. If $\{c_{H_j(F_i)} > 0,\ \forall 1 \leq j \leq K\}$, the file is considered to be in the bloom filter with probability ϵ. The expected false positive rate is $\epsilon = 2^{-\frac{m}{n}\cdot ln2}$, where m is the size of the bloom filter array, and n is the number of files that have fed into the bloom filter. We call this bloom filter is a matched bloom filter of file F_i.

The file information of bloom filters, denoted by 6-tuple representation $f = <\mathbf{v}, m_f, p, \overline{u}, N, n_i>$, needs to be periodically exchanged among neighbors. m_f denotes the number of files that have been fed into the bloom filter, p denotes the number of forwarding hops of the bloom filter, $\overline{u}$ denotes the average upload bandwidth of all nodes holding these fed files, N denotes the number of these nodes and n_i denotes the neighbor that forwards f. When node n_i initiates the bloom filter information of its own files, it needs to calculate the bloom filter of all files held by itself as $\mathbf{v}$, m_f which equals the number of files held by itself, and $\overline{u}$ which equals its own upload bandwidth. Then, it sends information $f = <\mathbf{v}, m_f, 1, \overline{u}, 1, n_i>$ to all its neighbors. In order to further reduce the network load of collecting file information, after a node receives the bloom filters from all its neighbors, it combines those with the same p and sends the combined bloom filter with $p \leftarrow p + 1$ to its neighbors. The node excludes the bloom filter from n_i from the combined bloom filter before sending it to n_i. Each node's initiated bloom filter has a TTL forwarding hop limit; that is, a node does not further forward bloom filters with p =TTL. When n_A receives the bloom filters from its neighbors n_B and n_C: $f_B = <\mathbf{v}_B, m_{(f,B)}, p_B, \overline{u}_B, N_B, n_B>$ and $f_C = <\mathbf{v}_C, m_{(f,C)}, p_C, \overline{u}_C, N_C, n_C>$ $(p_B = p_C)$, it combines them and the combination result is $f_{B+C} = <\mathbf{v}_B + \mathbf{v}_C, m_{(f,B)} + m_{(f,C)}, p_B + 1, \frac{(\overline{u}_B * N_B + \overline{u}_C * N_C)}{N_B + N_C}, N_B + N_C, n_A>$. The operation of $\mathbf{v}_B + \mathbf{v}_C$ adds the two counters of each array position to create a new bloom filter. For example, $\mathbf{v_B} = <7, 4, ..., 5>$, and $\mathbf{v_C} = <2, 4, ..., 1>$, then $\mathbf{v}_B + \mathbf{v}_C = <9, 8, .., 6>$.

Figure 4 shows an example of the bloom filter based proactive file information collection. Let us assume that the TTL equals to 2. We use a node's f_s^h to represent the bloom filter of files owned by node(s) s with h hops to the node. In the figure, n_A first receives the bloom filter from n_B, n_C and n_D, denoted as f_B^1, f_C^1 and f_D^1. Also, n_B receives f_A^1, f_E^1 and f_F^1. n_B then combines these bloom filters except f_A^1 and sends it to n_A, which is denoted by f_{E+F}^2. n_A also receives bloom filters from other nodes two hops away, i.e., f_{G+H}^2 and f_{I+J}^2. Therefore, after the information collection, each node has a set of bloom

1045-9219 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2327033, IEEE Transactions on Parallel and Distributed Systems

filters such as $S = \{f^1_B, f^1_E, f^1_C, f^2_{I+J}, f^2_{E+F}, f^2_{G+H}\}$ in n_A.

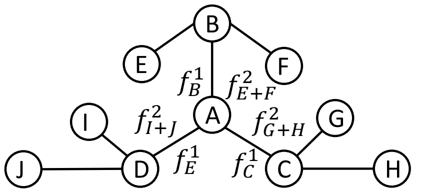


Fig. 4: Bloom filter based file information collection.

A matched bloom filer of a file indicates that this file is likely to have been fed into the bloom filter. The false positive rate ϵ can be constrained to a small value if the number of files fed into the bloom filter is limited. To achieve a given ϵ, the expected maximum number of files fed into the bloom filter is $-\frac{m \cdot \ln 2}{\log \epsilon}$. In file searching, since a file requester can only know the bloom filter file information of a group of nodes within TTL hops (e.g., f^2_{E+F} in Figure 4), the request needs to be forwarded to the neighbors of the matched bloom filter, then the neighbors of neighbors and so on until the requested file is located or the querying TTL is expired. Specifically, when a node initiates a file request, it searches its bloom filters to find a matched bloom filter of the requested file $f \in S$, and then sends the request to the node that forwarded the matched bloom filter (e.g., n_i in f). If the request receiver holds the requested file, it returns the file to the requester. Otherwise, it repeats the same process. That is, it tries to find a matched bloom filter $f' \in S'$ and then sends the request to n'_i in f'. Node n'_i continues the same process until the requested file is found or the querying TTL expires. If a node cannot find a matched bloom filter, it forwards the request to the supernode. We use *BloomFilter-noRank* to represent this searching method. For example, as shown in Figure 4, when n_A queries file F_i, n_A first checks all f in its S sequentially, and finds a matched bloom filter $\mathbf{v_{E+F}}$ in f^2_{E+F}. Then, n_A forwards the request to the forwarder of f^2_{E+F}, n_B. After n_B receives the query of F_i, it first checks whether it has the file. If not, it checks all f inside its S. n_B finds a matched bloom filter of F_i, $\mathbf{v_E}$ in f^1_E. It then forwards the request to n_E. If $TTL = 0$ and n_E does not have F_i, n_E forwards the query to the supernode.

4.5 Enhanced Bloom Filter based File Querying

We further improve *BloomFilter-noRank* by ordering the bloom filters in S in a certain order to find a matched bloom filter so that it is more accurate and faster to find file holders, and the located file holders have higher upload bandwidth to provide files. We denote this enhanced method by *BloomFilter*. In a bloom filter result f, a smaller m_f leads to a smaller false positive rate in file searching, hence higher file searching success rate. p indicates the searching path length. $\overline{u}$ indicates the node capability of serving the file requests. In order to enhance the file searching success rate and searching efficiency, after periodically receiving all bloom filter results from its neighbors, n_j ranks all bloom filter results by m_f, p and $\overline{u}$. Specifically, node n_j sorts all its bloom filter results firstly by m_f in an ascending order, and then sorts those with the same m_f in an ascending order of p, and finally sorts the bloom filter results in a descending order of $\overline{u}$. In file searching, a node checks the sorted f in S to find matched bloom filter. For example, in Figure 4, n_A has sorted S as below:

$$\begin{aligned} S = <\, & f^1_B = <v_B, 10, 1, 10Mb/s, 1, n_B>, \\ & f^1_C = <v_C, 10, 1, 5Mb/s, 1, n_C>, ..., \\ & f^2_{I+J} = <v_{I+J}, 20, 2, 10Mb/s, 2, n_D>, ... > \end{aligned} \quad (1)$$

n_A searches a matched bloom filter in the top-down order, and finds f^1_B. Then, it sends the request to n_B in f^1_B. Though f^2_{I+J} is also a matched bloom filter, it has lower rank than f^1_B, so it is not identified.

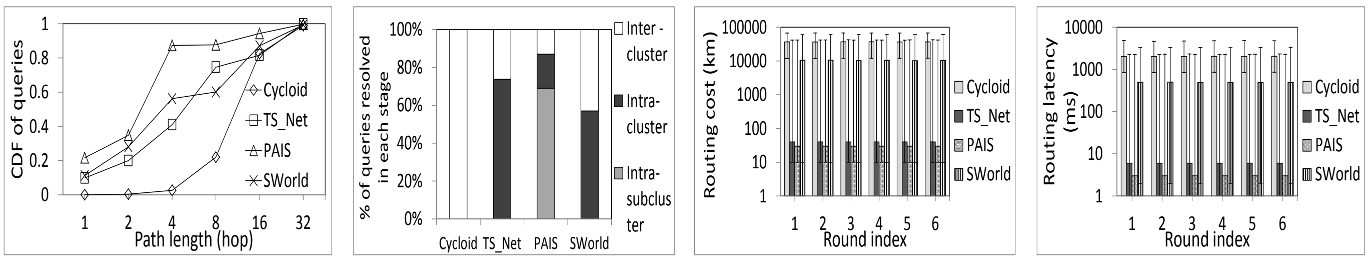
We assume that a node shares its previously visited files with other nodes. Thus, highly popular files have many replicas throughout the network, and the distributed intra-sub-interest-group file searching is efficient in searching highly popular files. However, it may not be easy to search unpopular files within TTL hops in a sub-interest overlay, since the replicas of such files are sparsely distributed over the network. The number of requests on files follows an exponential distribution over time [48]. Thus, the requests on unpopular files are limited, which can be handled by supernodes. Also, most of the requests on popular files are generated within a short period, which produces many file replicas during this time period. After a node receives a file, it enters this file into its own bloom filter and sends it to its neighbors in the next time period. Since a recently queried file is very likely to be queried again in a short time period, when a node initiates or receives a request, the requested file is very likely to be a newly added entry from the bloom filter from its neighbors. Therefore, to improve the searching accuracy, rather than checking an entire bloom filter, the node can first check the different part between the bloom filter in the previous time period and that in the current time period. We use *BloomFilter-PriorityCheck* to denote this method.

We use $\mathbf{v_T}$ and $\mathbf{v'_T}$ to denote the bloom filters having the same forwarder n_i and forwarding hops p received from the previous time period and current time period, respectively, by node n_j. $\mathbf{v_T}$ and $\mathbf{v'_T}$ represent the same group of nodes that p hops away from n_j. We use $\mathbf{v^-_T} = \mathbf{v'_T} - \mathbf{v_T}$ to represent the difference between $\mathbf{v'_T}$ and $\mathbf{v_T}$. For example, $\mathbf{v'_T} = [7, 4, ..., 5]$, and $\mathbf{v_T} = [2, 4, ..., 1]$, then $\mathbf{v^-_T} = \mathbf{v'_T} - \mathbf{v_T} = [5, 0, .., 4]$. The difference $\mathbf{v^-_T}$ is the bloom filter for newly added (i.e., requested) files among nodes p hops from node n_j. As recently queried files tend to be queried again and a smaller m_f value increases the searching accuracy, when n_j queries a file, it checks $\mathbf{v^-_T}$ first to improve the accuracy of searching. Specifically, the bloom filters of $\mathbf{v^-_T}$ are ranked based on m_f, p and $\overline{u}$ in the same way as previously introduced. The node checks the existence of the file in the sorted $\mathbf{v^-_T}$ first. If a matched bloom filter is found, the file searching is conducted in the same manner as in *BloomFilter*. Otherwise, the node uses the *BloomFilter* method, i.e., checking $\mathbf{v'_T}$.

5 Performance Evaluation

We implemented a prototype of PAIS on PlanetLab [53], a real-world distributed testbed, to measure the performance of PAIS in comparison with other P2P file sharing systems. We set the experiment environment according to the study results [48] of a BitTorrent trace [49]. We randomly selected 350 PlanetLab nodes all over the world. Among these nodes, we randomly selected 30 nodes as landmark nodes to calculate the Hilbert numbers of nodes. We clustered all nodes into 169 different locations

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2327033, IEEE Transactions on Parallel and Distributed Systems



(a) File search path length (b) Breakdown of querying stages (c) The routing distance (d) The routing latency

Fig. 5: The efficiency of file searching.

according to the closeness of their Hilbert numbers. We used the 56,076 files in the BitTorrent trace [49]. The number of interests in the system was set to 20, so we also set the dimension of the Cycloid DHT to 20. We simulated 100,000 peers by default in the experiments. Each peer was randomly assigned to a location cluster among all 169 clusters, and further randomly assigned to a PlanetLab node within this location. According to [48], a peer's requests mainly focus on around 20% of all of its interests. Thus, we randomly selected four interests (20% of total 20 interests) for each peer as its interests. The files are randomly assigned to a sub-cluster with the files' interest over the total 160 locations, and then randomly assigned to nodes in the sub-cluster. 80% of all queries of a requester target on files with owners within the same location, among which 70% of its queries are in the interests of the requester [39]. According to [48], 80% of all requests from a peer focus on its interests, and each of other requests is in a randomly selected interest outside of its interests. A request in an interest means a request for a randomly selected file in this interest. We also let each file have a copy in another peer in a different location in order to test the proximity-aware file searching performance.

Unless otherwise specified, there is only one group inside each sub-cluster. Inside each sub-interest group, the number of parents of a peer was set to $M = 3$. There were two node classes (i.e., Class1 and Class2) based on node bandwidth measured by the number of requests that can be handled per second. We used a bounded Pareto distribution [54] with shape 2 to generate node bandwidth capacity. The lower bound was set to 10 and 100, and the upper bound was set to 1 and 10 for Class1 and Class2, respectively. Each node is assigned to Class1 with a probability 15%, and to Class2 with a probability 85%. The TTL of intra-group searching was set to 2, considering that a file can be discovered within 2 hops on average in a common-interest node cluster [39]. Unless otherwise specified, PAIS uses *BloomFilter* for the intra-cluster searching. For the counting bloom filter [52], we set the false positive rate to 1% and the expected maximum number of inserted elements to 10,000.

Each experiment lasted six rounds. In each round, peers generated queries in turn at the rate of six queries per second in the system until each peer finished one query. In order to show the effectiveness of PAIS's proximity/interest clustering and searching method, we compared PAIS with SWorld [39], TS_Net [34] and Cycloid [7]. SWorld is an interest-aware clustering P2P system, which groups common-interest peers into a cluster. Inside a cluster, each peer randomly selects 20 peers as neighbors in order to reach all peers in the cluster within 4 hops. For an intra-cluster query, each peer chooses 10 randomly selected neighbors to forward messages with TTL=4. TS_Net is a proximity-aware clustering P2P system, which groups proximity-close peers into a cluster. Peers inside a cluster form a three-ary proximity-aware tree. In intra-cluster searching, a peer forwards a message through the tree structure within 7 hops in order to reach most other peers inside a cluster. In order to make these systems comparable to PAIS and guarantee a successful response to each query, we modified SWorld and TS_Net by using PAIS's DHT overlay for file searching whenever the intra-cluster searching fails. The DHT consists of supernodes and files are re-distributed among the supernodes and can be located by the `Lookup()` function. In SWorld and TS_Net, each cluster randomly selects 169 and 20 peers, respectively, to form the Cycloid overlay. We use Cycloid to denote the system without clustering, which always uses the DHT searching function `Lookup()` to find files.

5.1 The Efficiency of File Searching

Figure 5(a) shows the CDF of file queries versus query path length in hops of all systems. It shows that PAIS has more queries resolved within a small number of hops than other systems. It has 31%, 46% and 85% more queries resolved within four hops than SWorld, TS_Net and Cycloid, respectively. This is because PAIS's simultaneous proximity/interest-aware clustering enables nodes to find their interested files in proximity-close nodes in intra-cluster searching, which reduces the inter-cluster queries. Our experimental results show that the percent of requests resolved in intra-cluster searching are 81%, 73%, 57% and 0% in PAIS, TS_Net, SWorld and Cycloid, respectively. More inter-cluster queries lead to long query path lengths. PAIS also produces shorter query path lengths than other systems in intra-cluster searching. By clustering common-interest nodes, SWorld enables nodes to find their interested file in their cluster. However, the file providers may be unreachable from requesters within TTL. Therefore, queries in PAIS have shorter query paths than SWorld. Because PAIS further considers interest-based clustering compared to TS_Net, nodes can more quickly find interested files. Thus, PAIS leads to shorter path lengths than TS_Net. Cycloid only depends on DHT routing to find files without considering proximity and interests. DHT lookup path length is much longer than the path length of intra-cluster searching in the other systems. Therefore, Cycloid produces longer path lengths than other systems. TS_Net has more queries resolved within 8 and fewer queries resolved within 4 than SWorld. The intra-cluster forwarding TTL is 7 and 4 in TS_Net and SWorld, respectively. Therefore, TS_Net resolves more queries within 8 hops. The smaller percentage of queries within 4 hops in TS_Net than in SWorld is caused by TS_Net's smaller number of neighbors per node in a cluster for query forwarding than SWorld, which generates fewer queries resolved within short path lengths. This figure shows that PAIS

1045-9219 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2327033, IEEE Transactions on Parallel and Distributed Systems

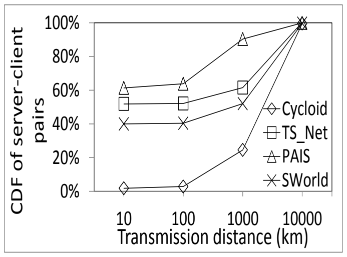
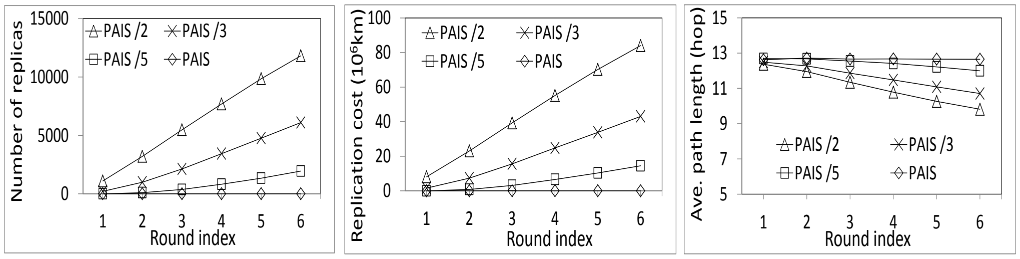


Fig. 6: CDF of server-client pair distance.



(a) Number of replicas (b) Replication cost (c) Path length

Fig. 7: The effectiveness of the file replication algorithm on file searching efficiency.

generates shorter path lengths than other systems, which indicates its high searching efficiency.

Figure 5(b) shows the percentage of queries resolved in each stage of searching including inter-cluster, intra-cluster and intra-sub-cluster searching in different systems. The inter-cluster stage means the `Lookup()` searching in the Cycloid DHT overlay. Cycloid only has inter-cluster searching, and TS_Net and SWorld do not have intra-sub-cluster searching. From the figure, we find that the percentage of requests resolved inside a cluster follows PAIS>TS_Net>SWorld>Cycloid=0 due to the same reasons as in Figure 5(a). We also find that most of the queries resolved inside a cluster are resolved within sub-cluster in PAIS. That is because most of the queries of a peer focus on its interests. This figure indicates the effectiveness of our intra-sub-cluster and intra-cluster searching, and it also verifies the effectiveness of proximity/interest-aware clustering in PAIS.

Figure 5(c) and Figure 5(d) show the median, the 5^{th} percentile and 95^{th} percentile of the routing cost measured by routing distance in km, and the latency of a request, respectively, in each round. We see that the median of both cost and latency of all systems follow PAIS<TS_Net<SWorld<Cycloid. Due to the same reasons as in Figure 5(a), PAIS has the smallest cost and latency among all systems, and Cycloid has the largest cost and latency. Because of the proximity-aware clustering in TS_Net, its intra-cluster searching is between physically close peers, which introduces smaller path distance and latency than those in SWorld. We can also see that the 5^{th} percentiles of routing cost and latency of Cycloid are much larger than those in other systems. This is because Cycloid only uses DHT routing without proximity or interest consideration, which generates long routing cost and latency, while in other systems, intra-cluster searching can efficiently resolve a large percentage of queries in small path lengths. These experimental results indicate the effectiveness of the proximity/interest clustering method in PAIS, which enables it to resolve queries with smaller cost and latency than other systems.

Figure 6 shows the CDF of server-client pairs over distance, which indicates the file transmission load from the server to the client hence the file sharing efficiency. The figure indicates that the expected distance of server-client pairs follows Cycloid>SWorld>TS_Net>PAIS. Both PAIS and TS_Net cluster peers with proximity-awareness and resolve most of queries within clusters. Thus, they can resolve more queries within shorter distances than other two systems without proximity-awareness. Compared to TS_Net, PAIS resolves more queries in intra-cluster searching. PAIS's proximity/interest clustering helps nodes quickly find their interested file in short distances. With only proximity-aware clustering, TS_Net allows nodes to search files within short distances but cannot guarantee fast file location. Therefore, PAIS generates a shorter server-client distance than TS_Net. In SWorld, a requester forwards its query to its neighbors, thus the file holders nearby respond the requester more quickly than the file holders far away. As a result, SWorld can find the files nearby with a higher probability than Cycloid, leading to shorter client-server distances than Cycloid. This figure indicates PAIS's higher file searching efficiency as it has shorter server-client distances than other systems.

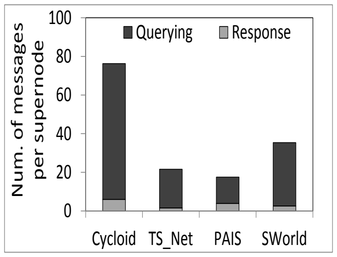
5.2 Performance of File Replication

In this experiment, when peers in one location request a file not existing in the location for at least t times, the file is replicated in their location. In the figures, PAIS/x denotes PAIS with the replication algorithm when $t = x$. In this experiment, there was a single replica for each file initially. Figure 7(a) shows the number of created replicas of PAIS with and without the replication algorithm with varying t values over the continuous six rounds. It shows that the number of created replicas follows PAIS<PAIS/5<PAIS/3<PAIS/2. This is because with the same file requests, a smaller t leads to more generated replicas. PAIS/2 has created nearly twice of the number of replicas of PAIS/3. Figure 7(b) demonstrates the replication cost measured by the sum of the distances between each pair of the original file owner and replica node. The replication cost also follows PAIS/5>PAIS/3>PAIS/2>PAIS, and the cost of PAIS/2 is about twice of PAIS/3 due to the same reason as in Figure 7(a). Both figures show that more replicas are created and hence more replication cost is generated as the experiment round increases because more file requests are generated.

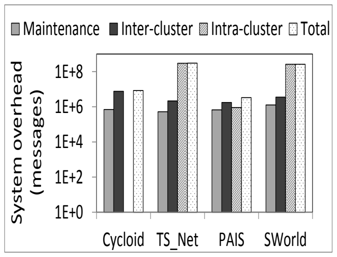
Figure 7(c) shows the average path length in each round of PAIS without and with the replication algorithm with varying t values over the continuous six rounds. First, we see that PAIS/x (x=2, 3, 5) generates shorter path lengths than PAIS, which shows the benefit of replicating frequently-visited files for nodes in a location. Second, we notice that the result follows PAIS/5>PAIS/3>PAIS/2, i.e., the path length decreases as the threshold t decreases. This is because a lower threshold leads to more replicas as shown in Figure 7(a), enabling nodes to find their requested files in their own locations. Third, we see that the path length decreases as time goes on. As more and more requests are sent out from nodes in a location, more replicas of different files are generated in their location, enabling nodes to retrieve files within one hop. We also find that the path length of PAIS/3 is slightly larger than PAIS/2. Considering the much larger replication cost and consistency maintenance cost of PAIS/2 over PAIS/3, $t = 3$ is a better choice to break the tie between replication/maintenance cost and query efficiency.

1045-9219 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2327033, IEEE Transactions on Parallel and Distributed Systems



(a) Avg. number of messages through a supernode



(b) Number of communication messages

Fig. 8: The overhead of file searching.

5.3 The Overhead of File Searching

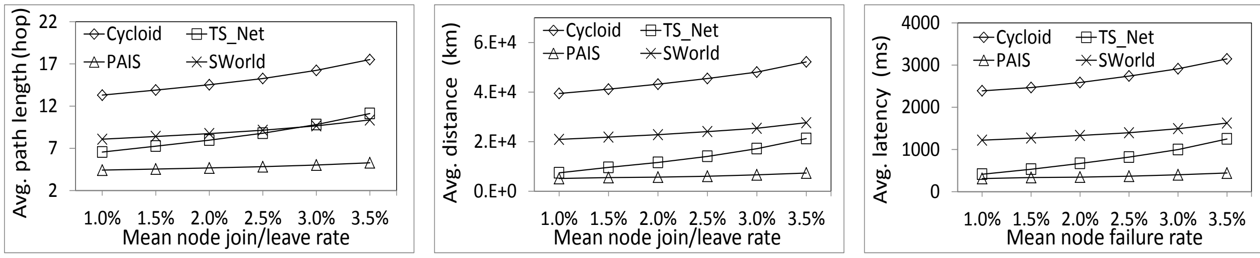
Figure 8(a) shows the average number of query messages traveling through supernodes per supernode in all six rounds, which reflects the load of supernodes in file querying. Cycloid does not use supernodes and it forms all nodes into a DHT, so we measured metrics on all nodes as its results. In PAIS, the supernodes handle the intra-interest group, intra-sub-cluster, inter-sub-cluster and DHT searching, while in other systems, the supernodes only handle the DHT searching. We measured the number of query messages forwarded to supernodes, and the response messages from supernodes. We see that the number of query messages per supernode follows Cycloid>SWorld>TS_Net>PAIS. Part of the reason for this result is that the percentage of queries resolved by DHT routing follows Cycloid>SWorld>TS_Net>PAIS as shown in Figure 5(b), and the `Lookup()` routing function introduces many routing messages. In PAIS, many queries are resolved by intra-cluster searching, which also introduces some query messages. However, such a query only produces one query message, which is much smaller than one DHT routing query. Therefore, PAIS still generates the smallest number of querying messages per supernode. The number of response messages per supernode follows Cycloid>PAIS>SWorld>TS_Net. PAIS has a larger number of response messages than TS_Net and SWorld, because it depends on supernodes to respond the requests after intra-interest group searching failures. Most queries are resolved in intra-cluster searching, so supernodes respond most queries. All queries are responded by supernodes by Cycloid, which leads to larger average response messages than PAIS. The number of response messages of SWorld and TS_Net follows the same order as the number of querying messages due to the same reason. Since a file query produces many searching messages but one response message if it is successfully resolved, the total number of messages per supernode still follows Cycloid>SWorld>TS_Net>PAIS. These experimental results indicate that PAIS has better performance than other systems in avoiding overloaded supernodes.

Figure 8(b) shows the number of communication messages for structure maintenance, inter-cluster querying and intra-cluster querying, respectively, and the total number of these messages in each system in all six rounds. The experimental results show that the number of maintenance messages follows SWorld>Cycloid>PAIS>TS_Net though it is not obvious in the figure. SWorld generates the largest number of maintenance messages, since each peer needs to maintain the connections to a large number of peers (i.e., 20) in a cluster. Cycloid generates a larger number of maintenance messages, because its larger Cycloid overlay with all peers participating in the DHT introduces much more messages to maintain the routing tables. PAIS has a smaller number of maintenance messages than Cycloid, because it needs to maintain a smaller Cycloid overlay, where each node is the supernode in charging of each sub-cluster. Also, due to its fine-grained interest and proximity clustering, each node maintains fewer neighbors in a sub-cluster than in a cluster in SWorld. TS_Net has the smallest number of maintenance messages, because each node only needs to maintain the connections to its 3 children and 1 parent inside its cluster. The order of the number of inter-cluster query messages of all systems is the same as that in Figure 8(a) due to the same reasons. The number of intra-cluster messages follows TS_Net>SWorld>PAIS. PAIS has much fewer intra-cluster query messages because of its limited hops in intra-cluster searching. TS_Net produces a larger number of intra-cluster messages due to its much longer path length (i.e., 7) in intra-cluster searching. Also, due to the much larger number of intra-cluster messages of SWorld and TS_Net, the total number of messages follows TS_Net>SWorld>Cycloid>PAIS. The figure indicates that PAIS introduces lighter overhead than other systems while achieves high file searching and sharing efficiency.

5.4 Performance of Dynamism Resilience

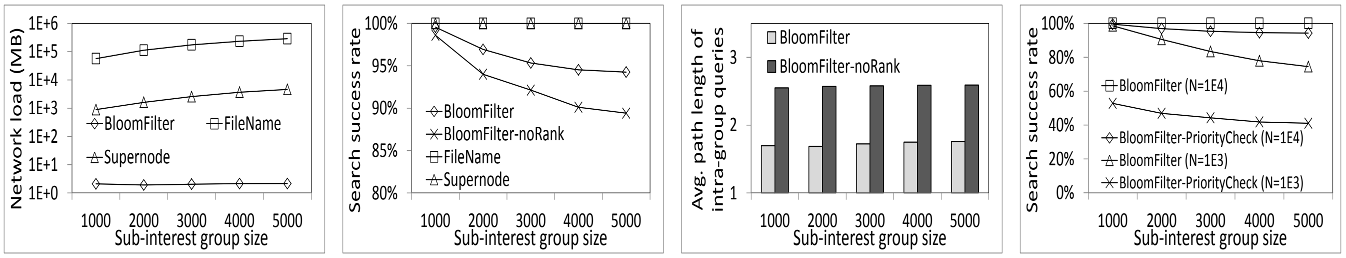
This experiment tests the performance of the dynamism-resilience of all systems. In this experiment, the peer failure rate followed a Poisson distribution [55], with the mean rate varying from 1% to 3.5% node failures per second. We measured the average performance of all six rounds. We used the average path length of all successful resolved queries as its path length, and used the maximum path length of DHT routing as the path length for a failed query. Figure 9(a) shows the average path length in hops of different systems with different mean node failure rates. It shows that the average path length follows Cycloid>SWorld≈TS_Net>PAIS due to the same reason as in Figure 5(a). TS_Net generates shorter path lengths than SWorld when the failure rate is low, because TS_Net resolves more queries inside a cluster than SWorld as shown in Figure 5(b), and the intra-cluster searching introduces much smaller expected path length than inter-cluster searching. The figure also shows that the average path length increases as the node failure rate increases for all methods. More peer failures lead to more querying failures, hence longer path lengths. From the figure, we can also see that the path length increase rate of all methods follows Cycloid>TS_Net>SWorld>PAIS. In the intra-cluster searching in PAIS and SWorld, when a node forwards a request to another node, if the receiver fails, the forwarder can choose another node to forward the request. TS_Net's single path leads to high failure rate, and then it resorts to DHT routing, which leads to longer path lengths. In Cycloid, a DHT routing failure leads to more routing hops. Therefore, PAIS and SWorld generate smaller increase of path length than TS_Net and Cycloid. PAIS has a larger percentage of queries resolved inside a cluster than SWorld, and the intra-cluster searching is much more dynamism-resilient than the inter-cluster searching. Thus, PAIS has a smaller increase rate than SWorld. TS_Net's intra-cluster searching leads to shorter path lengths than the DHT routing path lengths. Thus, TS_Net has a smaller path length increase rate than Cycloid. The figure indicates that PAIS can still generate the shortest query path length in node

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2327033, IEEE Transactions on Parallel and Distributed Systems



(a) Avg. path length (b) Avg. routing distance (c) Avg. routing latency

Fig. 9: File searching performance in node dynamism.



(a) Network load of proactive file information collection (b) Query success rate (c) Query path length (d) Query success rate of popular files

Fig. 10: Intra-sub-cluster file searching performance.

dynamism and the node dynamism has the smallest effect on PAIS's querying efficiency, which confirms its higher dynamism-resilience than other systems.

Figure 9(b) and Figure 9(c) show the average routing distance and latency of all systems, respectively. The figures show that the average routing distance and latency of all systems follow the same order as those in Figure 5(c) and Figure 5(d), respectively, due to the same reasons. The figures also show that all systems' routing distance and latency increase as the node failure rate increases, and the increase rate follows Cycloid>TS_Net>SWorld>PAIS due to the same reason as in Figure 9(a).

5.5 The Efficiency of Intra-sub-cluster Querying

In this experiment, we randomly selected one location out of all 169 locations, and used all PlanetLab nodes in this location to simulate a sub-interest group inside PAIS, where the number of peers was increased from 1,000 to 5,000 by 1,000 at each step. We randomly selected files to produce replicas and the number of replicas was 40 times of the number of peers inside this group. The TTL for file querying and information forwarding was set to 3. All peers inside the group only query files within 3 hops in order to test the file searching success rate solely determined by the false positive rate of the bloom filter method. We use *Supernode* to denote the method, in which all group peers forward the information of their owned files periodically to the supernode, and peers send file requests to the supernode. We measured the size of the transmitted information for proactive file information collection as network load.

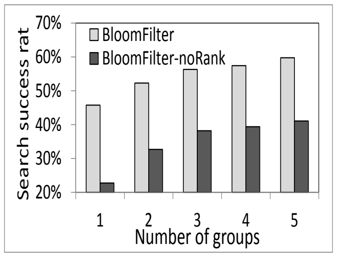
Figure 10(a) shows the average network load per node in one round. In *Supernode*, we measured the size of all file information transmitted to the supernodes as the result. The figure shows that the average network load follows *BloomFilter<Supernode<FileName*. By forwarding the file information to all nodes within 3 hops, *FileName* generates higher network load than *Supernode*. *BloomFilter* generates much lower network load than others. *BloomFilter* only needs to transmit the bloom filters by compressing the file information from many nodes into a single array of integers, which has a much smaller size than the combination of all file information. The figure indicates that the bloom filter based information collection is effective in reducing the network load.

Figure 10(b) shows the query success rate of intra-group searching without relying on the supernode in all methods. The figure shows that the success rate follows *FileName=Supernode>BloomFilter>BloomFilter-noRank*. *Supernode* has the file information of all peers in a group, so it generates 100% query success rate. In *FileName*, each node has each file's information associated with the file holders within TTL hops. Thus, it can definitely find the requested files within TTL hops. However, they introduce larger network load than *BloomFilter* as shown in Figure 10(a). *BloomFilter* has a higher success rate than *BloomFilter-noRank*, because its identified matched bloom filter has a smaller false positive rate. The figure indicates that *BloomFilter* is effective in searching files with a high query success rate while reduces network load. We also see that the success rates of both *BloomFilter* and *BloomFilter-noRank* decrease as the sub-interest group size increases. This is because more nodes inside a group mean more different files within TTL hops of a node, which leads to a larger false positive rate of the bloom filters in file searching.

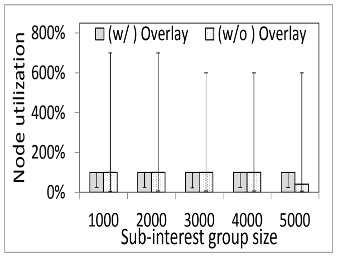
Figure 10(c) shows the average query path length of *BloomFilter* and *BloomFilter-noRank* inside the sub-interest group. Since *Supernode* does not have distributed intra-group file searching and *FileName* can always find the file holder, we did not measure their performance. From the figure, we can see that the average query path length of *BloomFilter* is shorter than that of *BloomFilter-noRank*. That is because *BloomFilter* gives the bloom filters with shorter path length a higher rank in choosing a matched bloom filter, while *BloomFilter* does not. This figure confirms the effectiveness of the enhanced bloom filter based file querying.

We use N to denote the maximum number of files inserted into the bloom filter in order to achieve ϵ. In this experiment, the percent of popular files was set to 1/10 of all files. All peers requested a randomly selected popular file in each round. As shown in Figure 10(d),

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2327033, IEEE Transactions on Parallel and Distributed Systems



(a) Searching success rate with sub-interest clustering



(b) Node utilization with node capacity classification

Fig. 11: Performance of intra-sub-cluster overlay.

we can see for both $N = 1,000$ and $N = 10,000$, *BloomFilter-PriorityCheck* produces a large query success rate than *BloomFilter*. The difference of two consecutive bloom filters only involves the popular files which are newly created. By checking this difference, *BloomFilter-PriorityCheck* produces a smaller false positive rate due to the smaller number of files involving in the bloom filter. The figure indicates that the effectiveness of *BloomFilter-PriorityCheck* in enhancing the success rate of *BloomFilter* in searching popular files.

5.6 Performance of Intra-sub-cluster Overlay

In this experiment, we used the same experimental environment as in Section 5.5, with the number of peers inside a sub-cluster set to 5000. To measure the performance of effectiveness of the sub-interest clustering in Section 4.1, we increased the number of groups from 1 to 5. Each peer randomly joined a group, and randomly requested files owned by the users inside the same group. Figure 11(a) shows the intra-group search success rate of *BloomFilter* and *BloomFilter-noRank* versus the number of groups. From the figure, we can see that the search success rate of *BloomFilter* is much larger than *BloomFilter-noRank* as shown in Figure 10(b), due to the same reason. We can also see that the search success rate of both methods increase as the number of groups increases. With more groups (i.e., more fine-grained interest classification), there are fewer nodes as well as fewer files inside a group, which leads to a higher percentage of requests resolved in intra-group searching. This figure indicates that the sub-interest clustering improves the effectiveness of the intra-sub-cluster searching by increasing the search success rate.

To measure the performance of effectiveness of overlay construction inside a group based on node capacity in Section 4.2, we measured the node utilization by $u_t = W/C$, where W is the total number of requests handled in a second and C is the node capacity. We used *(w/)Overlay* to denote our node capacity-aware overlay, and *(w/o)Overlay* to denote the same method to build the overlay except that Class1 and Class2 randomly selected nodes. Figure 11(b) shows the 5^{th} percentile, median and the 95^{th} percentile of all nodes' 99^{th} utilization during the experiment. From the figure, we can see that *(w/o)Overlay* has a much larger 95^{th} percentile, and a lower 5^{th} percentile results. This is because the nodes in Class1 handle most queries, and the workload may be too heavy for a low-capacity node. High capacity nodes in Class2 may have few requests, which fail to fully utilize their capacity. *(w/)Overlay* considers node capacity in overlay construction, so that higher capacity nodes receive more requests and lower capacity nodes receive fewer requests. Therefore, *(w/o)Overlay* has a much larger deviation of the 99^{th} percentile node utilization. This figure indicates that *(w/)Overlay* is effective to balance load among all peers according to their capacities.

6 CONCLUSIONS

In recent years, to enhance file location efficiency in P2P systems, interest-clustered super-peer networks and proximity-clustered super-peer networks have been proposed. Although both strategies improve the performance of P2P systems, few works cluster peers based on both peer interest and physical proximity simultaneously. Moreover, it is harder to realize it in structured P2P systems due to their strictly defined topologies, although they have high efficiency of file location than unstructured P2Ps. In this paper, we introduce a proximity-aware and interest-clustered P2P file sharing system (PAIS) based on a structured P2P. It groups peers based on both interest and proximity by taking advantage of a hierarchical structure of a structured P2P. PAIS uses an intelligent file replication algorithm that replicates a file frequently requested by physically close nodes near their physical location to enhance the file lookup efficiency. Finally, PAIS enhances the file searching efficiency among the proximity-close and common-interest nodes through a number of approaches. The trace-driven experimental results on PlanetLab demonstrate the efficiency of PAIS in comparison with other P2P file sharing systems. It dramatically reduces the overhead and yields significant improvements in file location efficiency even in node dynamism. Also, the experimental results show the effectiveness of the approaches for improving file searching efficiency among the proximity-close and common-interest nodes.

ACKNOWLEDGEMENTS

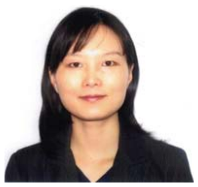
This research was supported in part by U.S. NSF grants IIS-1354123, CNS-1254006, CNS-1249603, CNS-1049947, CNS-0917056 and CNS-1025652, Microsoft Research Faculty Fellowship 8300751, and the United States Department of Defense 238866. An early version of this work was presented in the Proceedings of CCGRID 2009 [56].

REFERENCES

[1] BitTorrent. http://www.bittorrent.com/ [accessed in March 2014].
[2] Gnutella home page. http://www.gnutella.com [accessed in March 2014].
[3] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Proc. of the International Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, 2001.
[4] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *TON*, 2003.
[5] A. Rowstron and P. Druschel. Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proc. of Middleware*, 2001.
[6] B. Y. Zhao, L. Huang, and et al. Tapestry: an infrastructure for fault-tolerant wide-area location and routing. *J-SAC*, 2004.
[7] H. Shen, C. Xu, and G. Chen. Cycloid: a scalable constant-degree P2P overlay network. *Performance Evaluation*, 2006.
[8] Z. Li, G. Xie, and Z. Li. Efficient and scalable consistency maintenance for heterogeneous peer-to-peer systems. *TPDS*, 2008.
[9] H. Shen and C.-Z. Xu. Hash-based proximity clustering for efficient load balancing in heterogeneous dht networks. *JPDC*, 2008.
[10] FastTrack. http://www.fasttrack.nu/index_int.html [accessed in March 2014].

1045-9219 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

[11] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *Proc. of INFOCOM*, 2002.
[12] M. Waldvogel and R. Rinaldi. Efficient topology-aware overlay network. In *Proc. of HotNets-I*, 2002.
[13] Y. Zhu and H. Shen. An efficient and scalable framework for content-based publish/subscribe systems. *PPNA*, 2008.
[14] C. Hang and K. C. Sia. Peer clustering and firework query model. In *Proc. of WWW*, 2003.
[15] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proc. of ICDCS*, 2002.
[16] W. Nejdl, W. Siberski, M. Wolpers, and C. Schmnitz. Routing and clustering in schema-based super peer networks. In *Proc. of IPTPS*, 2003.
[17] P. A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing: A vision. In *Proc. of WebDB*, 2002.
[18] A. Y. Halevy, Z. G. Ives, P. Mork, and I. Tatarinov. Piazza: Data management infrastructure for semantic web applications. In *Proc. of WWW*, 2003.
[19] K. Aberer, P. Cudrè-Mauroux, and M. Hauswirth. The chatty web: Emergent semantics through gossiping. In *Proc. of WWW*, 2003.
[20] Morpheus. http://www.musiccity.com. [accessed in March 2014].
[21] P. Garbacki, D. H. J. Epema, and M. van Steen. Optimizing Peer Relationships in a Super-Peer Network. In *Proc. of ICDCS*, 2007.
[22] B. Mitra, A. K. Dubey, S. Ghose, and N. Ganguly. How do superpeer networks emerge? In *Proc. of INFOCOM*, 2010.
[23] Q. Hofstatter, S. Zols, M. Michel, Z. Despotovic, and W. Kellerer. Chordella - A hierarchical peer-to-peer overlay implementation for heterogeneous, mobile environments. In *Proc. of P2P*, 2008.
[24] M. Sachez-Artigaz, P. Garcia-Lopez, and A. F. G. Skarmeta. On the feasibility of dynamic superpeer ratio maintenance. In *Proc. of P2P*, 2008.
[25] H. Liu, J. Thomas, and P. Khethavath. Moving Target with Load Balancing in P2P Cloud. In *Proc. of Cloud*, 2013.
[26] P. Garbacki, D. H. J. Epema, and Steen M. V. The Design and Evaluation of a Self-Organizing Superpeer Network. *TC*, 2010.
[27] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of ACM SIGCOMM*, pages 329–350, 2001.
[28] S. Genaud and C. Rattanapoka. Large-scale experiment of co-allocation strategies for peer-to-peer supercomputing in P2P-MPI. In *Proc. of IPDPS*, 2008.
[29] R. Cuevas, N. Laoutais, X. Yang, G. Siganos, and P. Rodriguez. BitTorrent Locality and Transit Traffic Reduction: When, Why and at What Cost? *TPDS*, 2013.
[30] C. Decker, R. Eidenbenz, and R. Wattenhofer. Exploring and Improving BitTorrent Topologies. In *Proc. of P2P*, 2013.
[31] Y. Liu, L. Guo, F. Li, and S. Chen. A case study of traffic locality in Interenet P2P live streaming systems. In *Proc. of ICDCS*, 2009.
[32] H. Shen and K. Hwang. Locality-preserving clustering and discover of wide-area grid resources. In *Proc. of ICDCS*, 2009.
[33] F. Lehrieder, S. Oechsner, T. Hossfeld, Z. Despotovic, W. Kellerer, and M. Michel. Can P2P-users benefit from locality-awareness? In *Proc. of P2P*, 2010.
[34] M. Yang and Y. Yang. An Efficient Hybrid Peer-to-Peer System for Distributed Data Sharing. *TC*, 2010.
[35] C. Gross, B. Richerzhagen, D. Stingl, J. Weber, D. Hausheer, and R. Steinmetz. GeoSwarm: A Multi-Aource Sownload Acheme for Peer-to-Peer Location-Based Aervices. In *Proc. of P2P*, 2013.
[36] M.P. Manzillo, L. Ciminiera, G. Marchetto, and F. Risso. CLOSER: A Collaborative Locality-Aware Overlay SERvice. *TPDS*, 2012.
[37] G. Ruffo and R. Schifanella. A peer-to-peer recommender system based on spontaneous affinities. *TOIT*, (1), 2009.
[38] K. Elkhiyaoui, D. Kato, K. Kunieda, K. Yamada, and P. Michiardi. A scalable interest-oriented peer-to-peer pub/sub network. In *Proc. of P2P*, 2009.
[39] A. Iamnitchi, M. Ripeanu, E. Santos-Neto, and I. Foster. The Small World of File Sharing. *TPDS*, 2011.
[40] X. Cheng and J. Liu. NetTube: exploring social networks for peer-to-peer short video sharing. In *Proc. of INFOCOM*, 2009.
[41] Z. Li and H. Shen. Social-P2P: Social network-based P2P file sharing system. In *Proc. of ICNP*, 2012.
[42] K. C. J. Lin, C. P. Wang, C. F. Chou, and L. Golubchik. Socionet:a social-based multimedia access system for unstructured p2p networks. *TPDS*, 2010.
[43] Y. Li, L. Shou, and K. L. Tan. CYBER: A Community-Based Search Engine. In *Proc. of P2P*, 2008.
[44] F. Liu, Y. Sun, B. Li, B. Li, and X. Zhang. FS2You: Peer-Assisted Semi-Persistent Online Hosting at a Large Scale. *TPDS*, 2010.
[45] F. Liu, Y. Sun, B. Li, and B. Li. Quota: Rationing Server Resources in Peer-Assisted Online Hosting Systems. In *Proc. of ICNP*, 2009.
[46] Y. Hua, H. Jiang, Y. Zhu, D. Feng, and L. Tian. Semantic-Aware Metadata Organization Paradigm in Next-Generation File Systems. *TPDS*, 2012.
[47] H. Chen, H. Jin, X. Luo, Y. Liu, T. Gu, K. Chen, and L. M. Ni. BloomCast: Efficient and Effective Full-Text Retrieval in Unstructured P2P Networks. *TPDS*, 2012.
[48] G. Liu, H. Shen, and L. Ward. An Efficient and Trustworthy P2P and Social Network Integrated File Sharing System. *TC*, 2014.
[49] BitTorrent User Activity Traces. http://www.cs.brown.edu/ pavlo/torrent/ [accessed in March 2014].
[50] Z. Xu and et al. Turning heterogeneity into an advantage in overlay routing. In *Proc. of INFOCOM*, 2003.
[51] T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmaier. Space filling curves and their use in geometric data structure. *Theoretical Computer Science*, 181(1):3–15, 1997.
[52] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood. Fast hash table lookup using extended bloom filter: an aid to network processing. In *Proc. of SIGCOMM*, 2005.
[53] PlanetLab. http://www.planet-lab.org/ [accessed in March 2014].
[54] K. Psounisa, P. M. Fernandezb, B. Prabhakarc, and F. Papadopoulosd. Systems with multiple servers under heavy-tailed workloads. *Performance Evaluation*, 2005.
[55] F. A. Haight. Handbook of the Poisson Distribution. *New York: John Wiley & Sons*, 1967.
[56] H. Shen. PAIS: A Proximity-aware Interest-clustered P2P File Sharing System. In *Proc. of CCGRID*, 2009. Best paper award nominee, 4/271.



Haiying Shen received the BS degree in Computer Science and Engineering from Tongji University, China in 2000, and the MS and Ph.D. degrees in Computer Engineering from Wayne State University in 2004 and 2006, respectively. She is currently an Associate Professor in the Department of Electrical and Computer Engineering at Clemson University. Her research interests include distributed computer systems and computer networks, with an emphasis on P2P and content delivery networks, mobile computing, wireless sensor networks, and cloud computing. She is a Microsoft Faculty Fellow of 2010, a senior member of the IEEE and a member of the ACM.

Guoxin Liu received the BS degree in BeiHang University 2006, and the MS degree in Institute of Software, Chinese Academy of Sciences 2009. He is currently a Ph.D. student in the Department of Electrical and Computer Engineering of Clemson University. His research interests include Peer-to-Peer, CDN and online social networks.

Lee Ward is a principal member of technical staff in the scalable systems computing department at Sandia National Laboratories. As an inveterate student of operating systems and file systems, his interests have provided the opportunity to make contributions in high performance, parallel file systems, IO libraries, hierarchical storage management and compute cluster integration/management systems.