I. J. Computer Network and Information Security, **2012, 3, 1-7**
Published Online April 2012 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2012.03.01

Distributed Algorithms for Improving Search Efficiency in P2P Overlays

Chittaranjan Hota[1], Vikram Nunia[1], Antti Ylä-Jääski[2]
[1]*Dept. of Computer Science and Information Systems, Birla Institute of Technology and Science-Pilani Hyderabad Campus, Hyderabad, AP, India, 500078*
[2]*Department of Computer Science& Engineering, Aalto University, Helsinki, Finland*
hota@bits-hyderabad.ac.in, vikram.nunia@gmail.com, Antti.Yla-Jaaski@tkk.fi

***Abstract*—** Peer-to-peer (P2P) overlay is a distributed application architecture in which peers share their resources. Peers are equally privileged, equipotent participants in the application. Several algorithms for enhancing P2P file searching have been proposed in the literature. In this paper, we have proposed a unique approach of reducing the P2P search complexity and improving search efficiency by using distributed algorithms. In our approach a peer mounts other popular peer's files and also replicates other popular files or critical files identified using a threshold value. Once a file is mounted, file access requests can be serviced by transparently retrieving the file and sending it to the requesting peer. Replication used in this work improves the file retrieval time by allowing parallel transfer. We have shown the performance analysis of our proposed approach which shows improvement in the search efficiency.

***Index Terms*—** Algorithm; File Sharing; Replication; Mounting; Bootstrap Peer.

I. INTRODUCTION

The peer to peer (P2P) paradigm had started becoming popular in the middle of 2000 amongst internet music lovers. Since then, due to its inherent positive characteristics, the term 'P2P' has become very popular amongst internet users, researchers and industries. A system is to be considered P2P if the elements that form the system share their resources in order to provide the designated services. The elements in the system provide both client and server services i.e., providing services to other elements and requesting services from other elements. An overlay network is a logical network at the application layer providing connectivity, routing and messaging amongst the addressable endpoints. They have their own topology different from underlying physical network. They have their way of routing messages with the help of the Internet. They have their own way of addressing the endpoints. Overlay networks are frequently used as a substrate for deploying new network services, or for providing a routing topology not available from the underlying physical network as shown in Fig.1.
P2P overlay networks are categorized as unstructured and structured. An unstructured P2P network is composed of peers joining the network with some loose rules, without any prior knowledge of the topology. Gnutella [1] and Kazaa [2] are examples of unstructured P2P overlay networks. In structured P2P overlay networks, network topology is tightly controlled and content is placed not at random peers but at specified locations that will make subsequent queries more efficient. Most of the structured P2P overlays are Distributed Hash Table (DHT) based. Content Addressable Network (CAN), Chord, and Pastry are some of the well known examples of structured P2P overlay networks.
In this work, we have developed a unique approach of reducing the P2P search complexity and improving search efficiency by using distributed algorithms mounting and replication. A peer which serves more number of peers (we call it as a popular peer) mounts heavily used files from other peers. This popular peer when contacted by other peers transparently retrieves the files and gives those to the requesting peer.

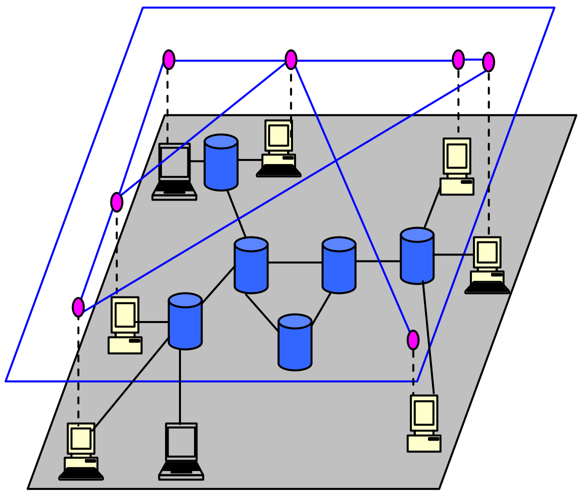


Figure1. An Example P2P Overlay Network

The replication algorithm developed replicates critical files at multiple peers. A critical file is a file that is accessed more than a threshold limit, which is tunable. The peers that are selected as location for storing replicas are judiciously decided on various factors like number of CPU, Storage capacity, and Peer connection degree at that peer.

The remainder of the paper is organized as follows. Section II reviews the related research. Section III discusses models and assumptions. Section IV presents our proposed implementation. In Section V, we describe our distributed algorithms. Section VI gives details about the simulation run. Finally, Section VII concludes the paper and presents future research directions.

II. RELATED WORK

Search in an unstructured peer-to-peer file sharing system is dependent on many factors centered on overlay topology, data placement and routing [3]. A good search mechanism is one which allows users to effectively locate desired data in a resource-efficient manner [3]. In unstructured overlays, there are several challenges like their large size, transitive population of nodes, heterogeneity, user autonomy etc. There are number of search algorithms proposed to meet these challenges. The first file sharing network [4] maintained a centralized index of all the files. Search is carried out by the central server itself without involving other nodes in the network. But central servers have the some limitations like single point of failure, scalability, etc. With litigations with Napster, Gnutella [1] became popular. Gnutella uses a fully decentralized search mechanism which is known as Flooding or, Breadth-first-search [5].Lv et al. [6] notes several limitations of flooding like heterogeneity issues, selecting appropriate TTL, handling of duplicate queries. Yang and Gracia-Molina[5] proposed flooding policy known as iterative deepening which performs multiple breadth-first searches with successively larger depths. But it still has disadvantages like huge response delay, handling of duplicate messages, etc.

Some schemes have used replicating objects in the network in order to increase efficiency and quality of results. Major considerations in replication are selection of objects and selection of sites. There are various replication techniques explored in literature like path replication, square root replication [6], Pull-Then-Push replication [7], etc.

Indexing is the most important tool for searching. Index building is about creating and maintaining data structures that have files and their location information. In local indices technique [8], each node maintains an index of data of all nodes within r hops of itself. When a node receives the query, it processes it on behalf of all the nodes within r hops of radius. Creating and maintaining such index involves extra overhead on the network. In routing indices technique [8], index is created for different topics in different routes. The index is used for choosing a neighbour to forward the query. In this technique, the aggregate updates are exchanged among the nodes to keep the index up to date. By sending only the aggregated vectors, the overhead is reduced. In attenuated bloom filter technique [9], an index for every neighbor and up to 'd' number of hops is maintained. In Zhang and Hu[10], a global but partial index is built using a Distributed Hash Tables (DHT) built on top of the unstructured overlay. In eSearch [11], index for every term is created. One node is responsible for maintaining an index of one term. Nodes analyze their documents and find top terms and publish them to the respective nodes responsible for those terms.

III. SYSTEM MODEL

A. *Requirements*

Sun Remote Procedure Call (RPC) API is used to facilitate communication amongst the peers. We have written the Interface Definition Language (IDL) specification for rpcgen compiler. In classic P2P overlay designs, participants agree on the protocol to be used, which in turn enforces the policy of resource allocation and usage amongst the peers. Peers donate their resources to a common pool which is then managed by overlay itself. On the other hand, the design proposed in this paper takes into account a peer-to-peer environment where:

- Peers have full control over their own files.
- A peer has to put all files which he or she wants to share in a shared folder.
- Any peer can read or update the other peer files. But he can't delete other peer's files.
- All peers have a client application and a server application. Before starting the client application, server application must be running.
- All peers know the central peer address. The central peer's address is provided through a file 'cn_addr.x'. This central node address is used for several purposes like bootstrapping etc.
- All peers are identified through their IP addresses.
- There is no limit on the number of peers in the network and also on the connection degree of the peers.
- Shared namespaces: In addition to sharing file contents, peers also share a common set of files.

B. *Network Layout*

We constructed P2P network having 25 peers. We used a centralized approach for connection setup and mounting files. Central peer (bootstrap peer) should be in running state all the time. Initially a peer knows only bootstrap peer address and from that bootstrap peer it will get other peers information which are available in the network, then it can connect to any number of peers which are available. Once it becomes the part of the overlay it can get all the services.

In the network layout shown in Fig. 3(a), DC means Directly Connected, CM means Connected through Mounting, and MFS means Mounted File System. Initially a node only knows the bootstrap node's address. First the node will notify the bootstrap node that it has got connected to the network. Then it will ask for available peers and choose whom to connect with directly as it's neighbor. After selecting the peers to whom it wants to connect it will send a *'friend'* message to the selected nodes. Friend message is a special message sent by a peer to notify other peers that "I am your neighbor and I am directly connected to you" so the other peers also update their connection information. Here connection information means to whom the peer is connected directly.

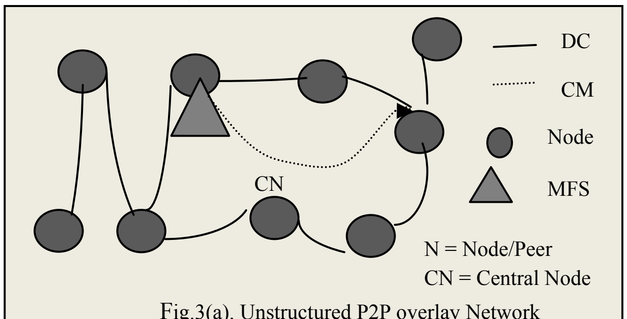


Fig.3(a). Unstructured P2P overlay Network

Central node is used to keep information about the network like, available peers in the network, selecting an appropriate peer for replicating files over it, selecting which peer file system peer wants to mount. When a peer finds a critical file, it asks the bootstrap peer to find out a suitable place for keeping the replica of this file. Central node will then ask every peer their resource information and compute *replication confidence (σ)* which is defined below. Then peer having maximum confidence will be selected for the file replication. Replication confidence is a value derived from the below expression:

$$\sigma= a*\alpha + b*\beta + c*\gamma + d*\delta \qquad (1)$$

Where, σ is replication confidence, α is access probability, β is storage capacity in TB, γ is connection degree, δ is main memory in GB, and a, b, c, d are tunable parameters which define the ratio of α, β, γ, δ to be used in calculation of σ respectively. In our setup we used a=0.4, b=0.3, c=0.2, d=0.1.

The α is calculated as the probability that the file to be replicated will be accessed by the peer. High probability means it will access the file more in future. The α is calculated by every peer 'i' as:

$$\alpha=(File\_access_i) / \sum_{i=1}^{i=n}(File\_access_i) \qquad (2)$$

Where File_access$_i$ is the number of accesses of the file to be replicated which is made by peer 'i' and 'n' is the total number of peers.

In file system mounting a peer will ask bootstrap node for available peers in the network and then can mount the selected peer's file system or specific file system. Specific file system means peer can select file type for mounting e.g., peer 'A' can mount only 'avi' files of peer 'B'. So a peer can either be connected directly or connected through mounting. But there is a difference in both the connections. If peer is connected directly then it will ask for a file and requested peer will search the file and respond back or forward the query. But if connected through mounting then peer will search the file directly in mounted file system and would not send request to any other peer for file if the file is found on mounted file system otherwise it will ask its neighbors for the file.

IV. IMPLEMENTATION ARCHITECTURE

Overall Structure

The core of our implementation is P2P client and server application which provides the handling of files, searching for files, with mounting and replication mechanisms.

Client Application

Client application is an interface for the user. The user can do several operations. On the invocation of client application, client application first will check if it is already connected or not. If connected already, it will ask user to reconnect to some other peers or to continue the same connections. If not connected or user wants to reconnect, then client will send request to server application through RPC call and in response will get all available peers. Then it will ask user to whom it wants to connect and send the request to the server that "I want to connect to the following peers". After connection establishment the user can search the files he or she wants. If the local file is searched, no request is sent to the server application. If shared or network files are searched then client will send request for same to the server. Server will respond with peer address at which the file is present. If file is not found, error is reported.

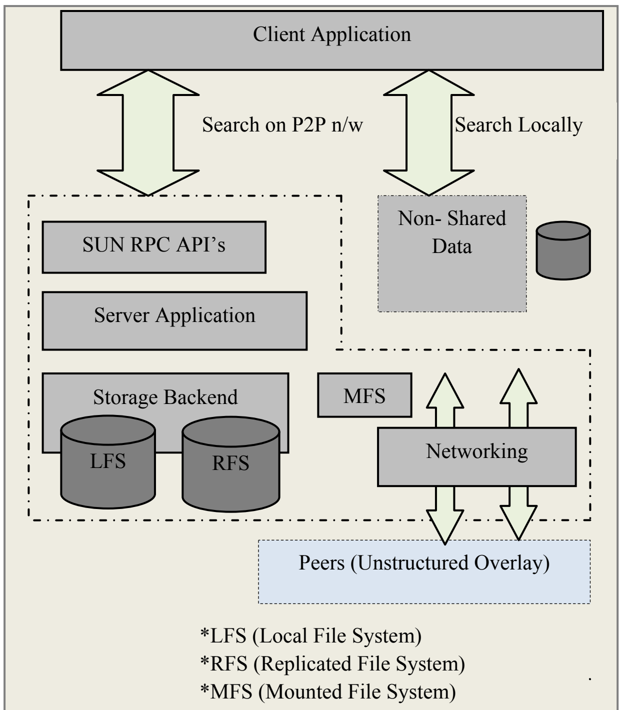


Fig.4(a) Architecture of peer-to-peer overlay network

If peer wants to mount a file system of any other peer then client application will send request for available peers to server application. Server will respond with the available peers. Client will select the peers and file type which it wants to mount. Client application then sends request for the same to server application.

Server Application

Server application runs in the background, does file replication automatically depending on criticality of the file, connection maintenance, etc. Implementation architecture of our proposed approach is shown in Fig.4(a).

V. DISTRIBUTED ALGORITHMS

We explain our approach using an example here. Consider a network with five peers only (for simplicity of explanation). Suppose, 172.16.5.7 is the IP address of the bootstrap node, which is in the running state. When BN's client application is invoked with no other peer being available then BN client application will connect to the server only. Now suppose another peer with IP address 172.16.5.5 enters into the network by invoking its server application. When client application will be invoked, only 172.16.5.7 is available and hence, 172.16.5.5 will choose it to connect. Later the peer with IP address 172.16.5.6 enters into the network. It will ask BN for available peers. Now available peers are 172.16.5.5 and 172.16.5.7. Suppose, the peer with IP 172.16.5.6 selects 172.16.5.5 as its neighbor and so on 172.16.5.4 and 172.16.5.3 connected to other peers as shown in Fig.5(a).

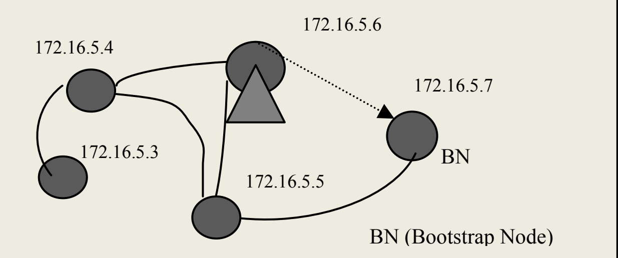
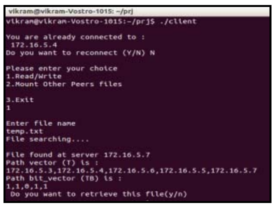


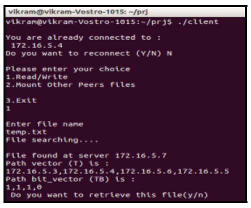
Fig.5 (a) Example P2P overlay Network

Suppose peer 172.16.5.3 wants to search a file *temp.txt* which is present on 172.16.5.7. Then 172.16.5.3 will search the file in its own storage and then in mounted file system if any. If not found then, send request for temp.txt to its neighbor 172.16.5.4 after adding the 172.16.5.4 to path vector (T) and corresponding bit as '1' in path bit vector (TB). The peer 172.16.5.4 will search the file temp.txt in its own storage, and if not found then sends the request to its neighbors 172.16.5.6 and 172.16.5.5 respectively. First 172.16.5.4 will update the T and TB and then 172.16.5.4 will send request to 172.16.5.6 after setting the corresponding TB entry to '1'. 172.16.5.6 will search the file in its own storage, if not found then it will check for the mounted file system. 172.16.5.6 has not mounted any file system so it will send an error 'File Not Found' to 172.16.5.4. The peer 172.16.5.6 will not send the request to 172.16.5.5 because path vector (T) received by 172.16.5.6 will contain 172.16.5.5 and corresponding path bit vector (TB) as '0'. Therefore 172.16.5.6 will think that file in not present on 172.16.5.5. Now 172.16.5.4 will set the bit to '0' corresponding to 172.16.5.6, set the bit to '1' corresponding to 172.16.5.5, and will send the request to 172.16.5.5, which will again run the same procedure but 172.16.5.5 will forward the request to 172.16.5.7 after updating T and TB because 172.16.5.7 was not present in the T received by 172.16.5.5. At 172.16.5.7 file is found in its storage so, 172.16.5.7 will return success to 172.16.5.5, 172.16.5.5 will return success to 172.16.5.4 and 172.16.5.4 will return success to 172.16.5.3.



```
vikram@vikram-Vostro-1015: ~/prj
vikram@vikram-Vostro-1015:~/prj$ ./client

You are already connected to :
 172.16.5.4
Do you want to reconnect (Y/N) N

Please enter your choice
1.Read/Write
2.Mount Other Peers files

3.Exit
1

Enter file name
temp.txt
File searching....

File found at server 172.16.5.7
Path vector (T) is :
172.16.5.3,172.16.5.4,172.16.5.6,172.16.5.5,172.16.5.7
Path bit_vector (TB) is :
1,1,0,1,1
 Do you want to retrieve this file(y/n)
```

Fig. 5(b) Path before mounting



```
vikram@vikram-Vostro-1015: ~/prj
vikram@vikram-Vostro-1015:~/prj$ ./client

You are already connected to :
 172.16.5.4
Do you want to reconnect (Y/N) N

Please enter your choice
1.Read/Write
2.Mount Other Peers files

3.Exit
1

Enter file name
temp.txt
File searching....

File found at server 172.16.5.7
Path vector (T) is :
172.16.5.3,172.16.5.4,172.16.5.6,172.16.5.5
Path bit_vector (TB) is :
1,1,1,0
 Do you want to retrieve this file(y/n)
```

Fig. 5(c) Path after mounting

```
F_Search(f, T,TB)
1.begin
2.      Traversed_path:T;        Traversed_path_bit_vector:TB;
        Self_addr:S; File_to_search:f; File Server:FS;
        Mount Server:MS; Neighbors:N; Result:R; Replica←0;
3.      Search file 'f' in shared folder
4.      if ('f' found ) then
5.              T←T U S; TB←TBU "1"; Flag←1;
6.              if( replicate(FS)) then
7.                  Replica←1;
8.              end
9.              return R←{T,TB,FS,Flag,Replica};
10          else
11.                             //search in mounted file structures
12.             if(Search(f,MFS)==true)then
13.                T←T U S; TB←TBU "1"; Flag←1;
14.               if( replicate(FS)) then
15.                 Replica←1;
16.               end
17.              return R←{T,TB,FS,Flag,Replica};
18.             else
19.              T←T U N;
20.               for each neighbor n in same order as in N
21.               do
22.                         TB←TB U "0";
23.               end
24.               for each neighbor 'i' in Ndo
25.                         TB(i)← "1";
26                          R=F_Search(f,T,TB) of i;
27.                         ifR.Flag=1then return R;
28.                         TB(i)←0;
29.               end
30.             end
31.         end
32.     return R=(T,TB,0,0);
33.end
```

Fig. 5(d) File Searching Algorithm

Finally 172.16.5.3 will get file path as 172.16.5.3 ← 172.16.5.4 ← 172.16.5.5 ← 172.16.5.7 as shown in Fig.5(b). Now, suppose peer 172.16.5.6 decides to mount **.txt* files of 172.16.5.7 then it will request 172.16.5.7 to give a list of all *.txt files and 172.16.5.6 will mount these files. Suppose 172.16.5.3 again request for same file *temp.txt* then when request will come to 172.16.5.6 from 172.16.5.4, 172.16.5.6 will find temp.txt in mounted file system of peer 172.16.5.7 and will respond directly to 172.16.5.4 that the file temp.txt is present on 172.16.5.7. Thus path obtained is 172.16.5.3 ← 172.16.5.4 ← 172.16.5.6, as shown in Fig.5(c) of our run. Through mounting path obtained is shorter and also time taken to locate the file is less. Hence our approach reduces the file searching time. Algorithm for file searching is shown in Fig.5(d).

```
file_retrieval(f,T,TB)
1. begin
2.     Searched_path:T; Searched_path_bit_vector:TB;
       File_to_retrieve:f;Result:R,R2;
3.     if(R.Replica= =1)then
4.         R2←find_path(R .replicated_peer);
5.     end
6.     if(R2.path_found==true) then
7.         retrieve_file_parallel(R,R2);
8.     else
9.         retrieve_file(R);
10.    end
11. end
```

Fig.5(e) File retrieval algorithm

```
replicate(f,callee)
1.  begin
2.      File Server: FS; Mount Server: MS; IP address:IP;
3.      if(callee = = FS)then
4.          file←file_replicated(f);
5.          file.no_of_accesses←file.no_of_accesses+1;
6.          if(file .replica= =false)then
7.              if(file.no_of_accesses>= THRESHOLD) then
8.                  file.replica_IP←best_peer();
9.                  replicate_file(file.replica_IP,f);
11.                 update(f,file);
12.                 return true;
13.             else
14.                 update(f,file);
15.                 return false;
16.             end
17.         else return true;
18.         end
19.     elsif (callee= = MS) then
20.         return replicate(f,FS);
21      end
22. end
```

Fig. 5(f) File replication algorithm

When same file is requested for retrieval by 172.16.5.3 then file will be retrieved through the path obtained and 172.16.5.6 will temporarily retrieve temp.x directly and send it to 172.16.5.4. After sending file to 172.16.5.4, 172.16.5.6 will delete that file. In this way mounting reduces the searching and retrieval time of files. Now suppose temp.txt is accessed more than the THRESHOLD limit then 172.16.5.7 will ask bootstrap (as in the case it is same node) node for appropriate peer to replicate the file. Bootstrap node will send a message to all peers to return their *replication confidence.* Then after getting all responses BN will select the peer with maximum replication confidence. Algorithm for retrieval and replication are given in Fig.5 (e) & Fig.5 (f).

Case1. Let 172.16.5.4 has maximum replication confidence. Then BN will send IP address 172.16.5.4 as best peer for replication to the requesting peer. Now 172.16.5.7 will replicate the file temp.txt on 172.16.5.4. Also the file server (172.16.5.7) will send a message to 172.16.5.6 which mounted *temp.txt* so that 172.16.5.6 knows that file is replicated on peer 172.16.5.4. Now if request comes for same file *temp.txt* then, peer 172.16.5.4 will respond directly to peer 172.16.5.3. Thus again reducing the search time and also retrieval time because path to retrieve that file will be shorter.
Case2. Let 172.16.5.3 has maximum replication confidence. Then time taken to search file *temp.txt* will be same as without replication but if file is asked for retrieval then 172.16.5.3 will retrieve the *temp.txt* through 172.16.5.6 (path will be x.3←x.4←x.6←x.7) and 172.16.5.5 (path will be x.3←x.4←x.5) parallely. Thus it reduces the file retrieval time.

VI. PERFORMANCE ANALYSIS

The proposed algorithms are implemented in C using RPC (Remote Procedure Call) API's and tested over twenty five machines.

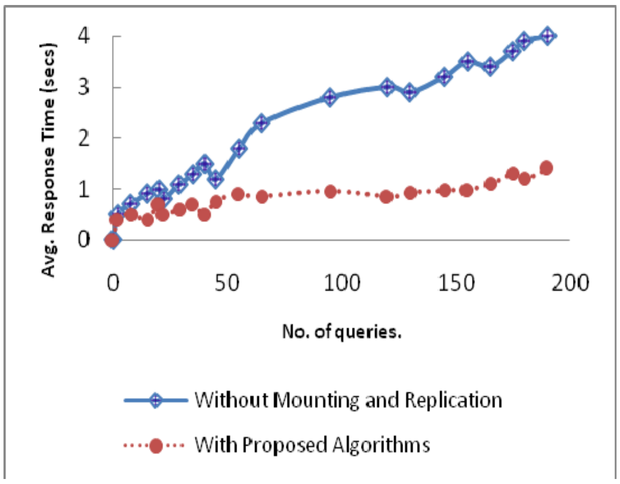


Fig.6 (a) Average response time measured in seconds with and without proposed algorithms.

To measure the search efficiency we used average response time, average file retrieval time and average packets sent by each node (average network traffic). Average response time is used to compare how long a user has to wait for file information. Average file retrieval time is used to compare how long a user has to wait for file after selecting it for download. Fig 6(a) shows that in our approach, average response time decreases. For more than 60% queries average response time decreases by 30% to 60%. The average file retrieval time also decreases with proposed algorithms as shown in the Fig. 6(b). The network traffic reduces with more replication and mounting as shown in the Fig. 6(c).

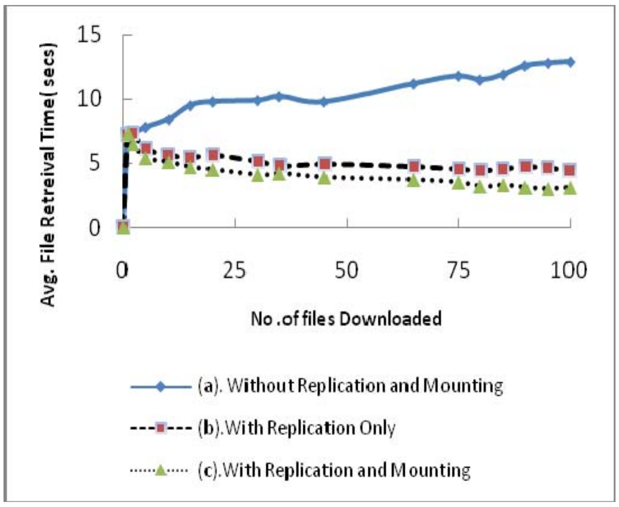


Fig.6 (b) Average file retrieval time measured in seconds.

In Fig. 6(b) we have considered three cases (a) Without mounting and replication, (b) No mounting but only replication (replication done automatically) and (c) With both mounting and replication. Initially the average time taken in case(b) will be the same as in case(a) but later, as the number of file accesses increase, which increases criticality of files thus, leading to replication of those files which result in fast retrieval of the files. In case (c) average time taken will be lesser than case (a) and case (b). For more than 70% of downloads average time taken decreases by 40% to 70%. In average file retrieval time all files downloaded were of 22.4 MB size (it was a video file). The same file was renamed with different index e.g., file1.avi, file2.avi, etc.

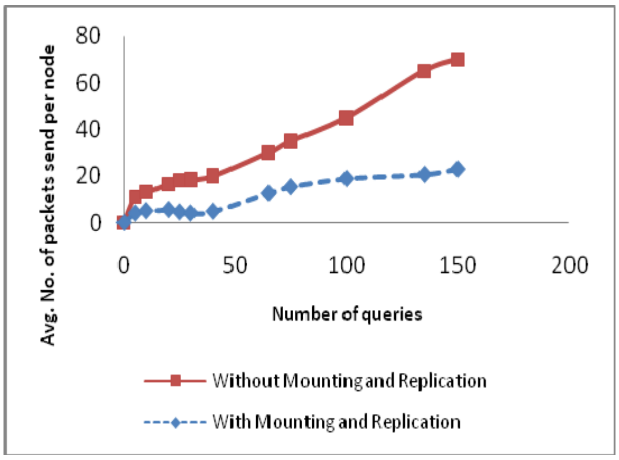


Fig.6(c) Average number of packets sends per node with and without Mounting and Replication.

For network traffic we have counted average number of packets sent by every node. We observed that after certain number of queries (after 60 queries) packet drop starts which increases network traffic but with proposed algorithms packet drops are less because path traveled is shorter than usual path and hence reduces network traffic. The resulting graph is shown below.

In Fig. 6(c), initially average number of packets with our approach is less. After 20 to 25 queries the average number of packets is again reduced because of replication. But after 40 to 45 number of queries average number of packets started increasing because of packet loss, network traffic, etc.

VII. CONCLUSION

The objective of searching and retrieval of a file is to reduce delay with minimal overhead. The proposed approach of mounting files and replication of critical files only reduces the average file searching and retrieval time. Hence improves the search efficiency. The results also show the same. Searching and retrieval time will be less because less number of hops are traversed resulting in less number of packets to be sent resulting in reduced network traffic and increased efficiency and throughput of the network as observed in our performance analysis section. Also, all the parameters used in the algorithms like parameters for replication confidence; threshold parameters etc. are tunable. These parameters can be tuned according to the network requirements and therefore, making this approach more flexible and dynamic. We plan to extend our work by making the P2P lookup more resilient by adding fault resilience into the search.

REFERENCES

[1] “The Gnutella Protocol Specification 0.6,” http://rfc-gnutella.sourceforge.net.

[2] “KaZaA,” http://www.kazaa.com.

[3] Neil Daswani, Hector Garcia-Molina and Beverly Yang, “Open Problems in Data-Sharing Peer-to-Peer Systems,” In Proceedings of the 9th International Conference on Database Theory (ICDT ’03), pp. 1–15, London, UK, 2002, Springer-Verlag.

[4] “Napster,” http://en.wikipedia.org/wiki/Napster.

[5] B. Yang and H. Garcia-Molina, “Improving search in peer-to-peer networks. In Distributed Computing Systems,” In Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS ’02), pp. 5–14, July 2002, IEEE Computer Society.

[6] Qin Lv, Pei Cao, Edith Cohen, Kai Li and Scott Shenker. “Search and replication in unstructured peer-to-peer networks,” In Proceedings of the 16th International Conference on Supercomputing (ICS ’02), pp. 84–95, New York, NY, USA, 2002, ACM Press.

[7] Elias Leontiadis, Vassilios V. Dimakopoulos and Evaggelia Pitoura. E., “Creating and Maintaining Replicas in Unstructured Peer-to-Peer Systems,” In 12th International Euro-Par Conference on Parallel Processing, pp. 1015-1025 Berlin, Heidelberg, 2006, Springer-Verlag.

[8] Arturo Crespo and Hector Garcia-Molina, “Routing Indices For Peer-to-Peer Systems,” In Proceedings of the

22nd International Conference on Distributed Computing Systems (ICDCS'02), pp. 23–34, Washington DC, USA, July 2002, IEEE Computer Society.
[9] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski,Patrick Eaton, Dennis Geels, RamakrishanGummadi, SeanRhea, Hakim Weatherspoon, Westley Weimer, Chris Wells and Ben Zhao, "An Architecture for Global-scale Persistent Storage," ACM Special Interest Group on Programming Languages (ACM SIGPLAN), Vol. 35, Issue 8, pp. 190–201, New York, NY, USA, Nov 2000, ACM Press.
[10] Rongmei Zhang and Y.C. Hu. "Assisted Peer-to-Peer Search with Partial Indexing," IEEE Transactions on Parallel and Distributed Systems, Vol. 18, Issue 8, pp. 1146–1158, Aug 2007, IEEE Computer Society.
[11] Chunqiang Tang and Sandhya Dwarkadas, "Hybrid Global-local Indexing for Effcient Peer-to-Peer Information Retrieval," In Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation, Vol. 1, pp. 16–16, Berkeley, CA, USA, 2004, USENIX Association Berkeley, CA, USA.

Chittaranjan Hota received the B.E in Computer Engineering, Amravati (MS), India in 1990, ME in Computer Science and Engineering from Thapar Institute of Engineering and Technology, Patiala India in 1998 and PhD in Computer Science and Engineering from BITS, Pilani India in 2006. He is currently Associate Professor and Head of the Department at the Department of Computer Science and Information Systems BITS, Pilani Hyderabad Campus, Hyderabad. His research interests include computer networks and distributed systems.

Vikram Nunia received B.Tech. from BK Birla Institute of Engineering and Technology, Pilani India in 2011. He is currently pursuing his Master of Engineering from BITS-Pilani Hyderabad Campus, Hyderabad. His research areas lie in the area of cloud computing, distributed systems, computer networks and algorithms.

Antti Yla-Jaaski is a Professor for the Data communications software laboratory (major and minor) since 2002 at the Department of Computer Science and Engineering, Aalto University, Helsinki Finland. He received his PhD from ETH Zurich in 1993. He has worked with Nokia from 1994-2004 in several research and research management positions with focus on future Internet, mobile networks, applications, services and service architectures. He was a Research Fellow on Internet Technologies in Nokia Research Center until 2008. In Data Communications Software, his expertise area is Internet technologies, application and service development in the Internet, and network architectures. Currently his primary research interests include Green ICT, energy efficient communications and computing, cloud computing, massive scale machine-to-machine communication systems and Internet of Things.