



History:

Received: September 4, 2017
Accepted: December 15, 2017
First Published:
Collection year: 2017
Confirmation of publication: Accepted

Identifiers and Pagination:

Year: 2017
Volume: 2
First Page: 4
Last Page: 14
Publisher ID: [10.21065/2520-596X.2.4](https://doi.org/10.21065/2520-596X.2.4)
DOI: <http://dx.doi.org/10.21065/2520-596X.2.4>

Corresponding author:

Fatma Elarabi, Department of
Mathematics, Suez Canal University,
Ismailia 41522, Egypt.
fatmaarabi.93@gmail.com

Citation:

Fatma Elarabi, Aya Aboelkhair, Alaa
Mohammed, Ehab Morsy. Storage
algorithm for unstructured peer-to-
peer networks. 2017 Adv Cal Anal,
Vol. 2, p 4-14

Review Article

STORAGE ALGORITHM FOR UNSTRUCTURED PEER-TO-PEER NETWORKS

Fatma Elarabi, Aya Aboelkhair, Alaa Mohammed, Ehab Morsy

Department of Mathematics, Suez Canal University, Ismailia 41522, Egypt

Abstract

In this paper, we consider the problem of storage and search in peer-to-peer networks. It is well known that all storage systems try to meet two conflict goals, increasing the success rate within reasonable response time and decreasing the worst case message complexity. In this result, we focus on designing an efficient algorithms for storage and search in unstructured peer-to-peer networks. In the storage, we present an algorithm for storage in which we try to maintain a reasonable number of copies of each file such that these copies can be reached in almost the same time from all nodes. In the search algorithm, we initiate a reasonable number of messages from the node that requests the file such that they cover random parts of the system. We evaluate our algorithm by applying it to random networks with various sizes.

Keywords: Distributed Systems, Distributed Algorithms, Peer-to-Peer Net-works, Distributed Storage Systems.

Correspondence: **Fatma Elarabi**, Department of Mathematics, Suez Canal University, Ismailia 41522, Egypt. fatmaarabi.93@gmail.com

1 Introduction

In this section we present basic definitions of computer networks, distributed systems, and their applications.

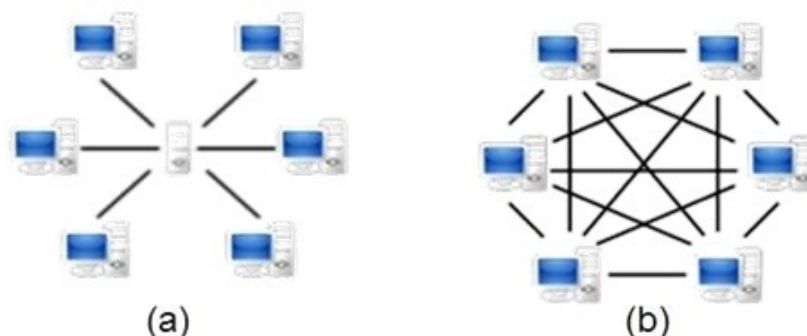


Figure 1: (a) Client/Server Network Systems, (b) P2P Network Systems

A network is a group of computers and associated devices that are connected by communications facilities, and are capable of exchanging information, software, and

Editorship:

Zhilin Li, Ph.D. (Applied Mathematics) Professor of Mathematics, Department of Mathematics, North Carolina State University, Raleigh, NC 27695-8205 USA

Funding:

The author is not supported by any organization/ institution.

Competing Interests:

The authors declare no competing interests

Additional information is available at the end of the article.

hardware resources among themselves (Tanenbaum, 2011). Clearly, sharing resources in networks reduces the cost and exchanges information very fast. There are different geographic types of networks. Personal Area Network (PAN) is a computer network used for data transmission among devices such as computers, telephones and personal devices. Local Area Network (LAN) is a computer network that interconnects computers within a limited area such as a school, computer laboratory or office building. Metropolitan Area Network (MAN) is a large computer network that spans a metropolitan area or campus. Wide Area Network (WAN) is a network that covers a broad area using leased telecommunication lines. Internetworking (simply, internet) is the practice of connecting a computer network with other networks through gateways that provide a method of routing packets between the networks.

There exist two common management methods of networks, client-server and peer-to-peer. A Client-server network is a computer network that includes one centralized device (called the server) to which many other devices (called the clients) are connected. The clients run applications, programs, and access data that are stored on the server. A peer-to-peer network is a network where the computers act as both clients and servers; there is no assigned role for any peer and each of them usually runs a similar software.

What is Distributed System?

A distributed system is a computer network that consists of autonomous components and its users deal with it as a single system. That is, distributed systems have independent devices and hence they are easy to expand. Distributed systems are often organized by a layer of software that is logically placed between a higher-level layer consisting of users and applications, and a layer underneath consisting of operating systems and basic communication facilities, as shown in Figure 9 (Tanenbaum & Van Steen, 2007). Such a distributed system is called mid-dleware. The middleware layer runs on all machines, and offers a uniform interface to the system

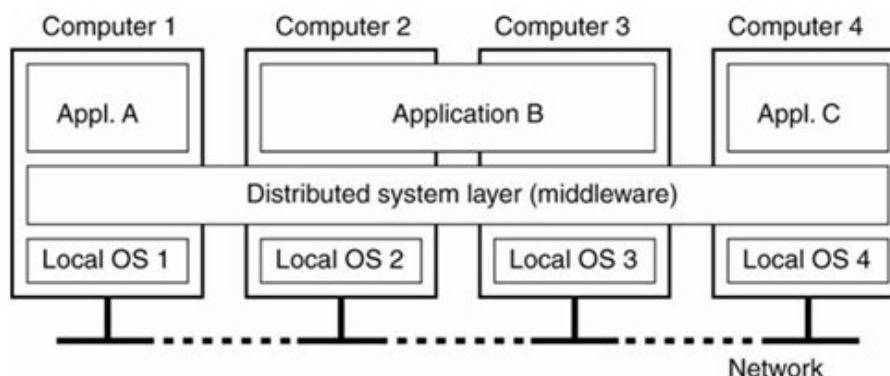


Figure 2: A distributed system organized as middleware.

Note that, a group of devices attains a less cost, more computing power, and a better performance than mainframes. In addition, if a device crashes in a distributed system, the system can still survive. Also, computing power can be increased by adding new devices to the system. As mentioned above, peer-to-peer network is a type of architecture in which nodes are connected with each other and share resources with each other without the central server. Bittorrent and Freenet are two of the well known applications of peer-to-peer systems that are used in file sharing between different peers. There are two types of Peer-to-peer systems, structured and unstructured systems. In a structured Peer-to-peer architecture, the overlay network is constructed using a deterministic procedure. Most peer-to-peer systems use distributed hash table (DHT) to organize processes between different peers of the system; when looking up for a data item, the address of the node

containing that data is returned from the DHT. Unstructured peer-to-peer systems rely on randomized algorithms for constructing and controlling the network. Each peer collects information about its neighbors. Data items are randomly placed on nodes. When a node looks up for a specific data item, it prepares and sends a search query through the network.

The most common types of communication in peer-to-peer systems are flooding and random walk. Flooding is a simple to implement routing technique in computer networks in which a node can transmit packets to all its neighbor nodes in the network. Note that, packets might be delivered multiple times to the same node. Specific procedures can be applied to prevent duplicating packets such as using a hop count or a time to live count and associate it with each packet, or allowing each node to keep track of every packet and only resend each packet once. Clearly, flooding is costly in terms of wasted bandwidth.

Random Walk technique is a simple routing technique in computer network in which a node chooses one peer from its neighbors randomly to send the packet. It is well known that a random walk covers the whole network in almost linear time in the input size of the network, with high probability. Moreover, the covering time can be reduced by initiating several independent random walks. On the other hand, there is a nonzero probability that it attains the worst performance.

The rest of this paper is organized as follows. The next section presents previous work on storage and search problem in peer-to-peer systems. Sections 3 and 4 presents our algorithm and the experimental results, respectively. The paper concludes with a summary of the results and suggestions for future research in Section 5.

2 Related Work

In this section, we present previous work that related to the underlying problem. We start with results on peer-to-peer centralized file sharing systems.

2.1 Centralized Systems

Centralized file sharing uses a central file server to maintain locations of stored files. To get to the files you first have to access the central file server.

Napster and Bittorrent are two applications of the centralized file sharing systems.

BitTorrent: When someone (the client) looks for a file, the client sends a request through peer-to-peer communication. It may get the file in pieces from different computers of the system. All peers cooperate in uploading and downloading parts of files. Peers downloaded a file are encouraged to stay online for uploading the file to other peers in the network.

Napster: A central server manages and control all processes in the network. Each peer is connected to the central server and send it data items. When a user looks for a file, he queries the Napster Index, which directs them to the computer that has the file. A separate connection is made between the two peers and the transmission of the requested file is initiated.

2.2 Decentralized Systems

2.2.1 Structured Systems

Chan, Ho, Shih, and Chung (2010) proposed a peer-to-peer storage system, called Malugo, which introduces a new replication technique. They designed efficient file cache and replication algorithms in order to attains a reliable file sharing system. Their system consists of three modules, file management, intra-overlay, and inter-overlay. The file management module is responsible for file insertion, retrieve, recovery, replicate, and cache from a storage peer. The inter-overlay module provides the ability for communication among groups of peers. The intra-overlay module provides a procedure of

locating peers to the suitable groups of the Chord system. Malugo is based on Chord protocol but has some improvements on locality, replication, and load balancing techniques. Assume that a new coming node N needs to join the system. If the suitable group is not close enough, node N forms a new group, and then N copies data back from neighbor groups. Otherwise, N joins the suitable group with Chord protocol, and then copies data back from previous node. When a peer uploads a file, it notifies the root of its group. When a peer looks for a file, it asks its root to request the file according Chord protocol. The algorithm records the download frequency of each file of each peer, increases its download rate, and then replicates the file to its predecessor peer if the rate exceeds a predefined upper bound. Jin, Wang, and Chen (2005). presented an efficient replica location technique in peer-to-peer system, called Boundary Chord, that is formed from multiple rings.

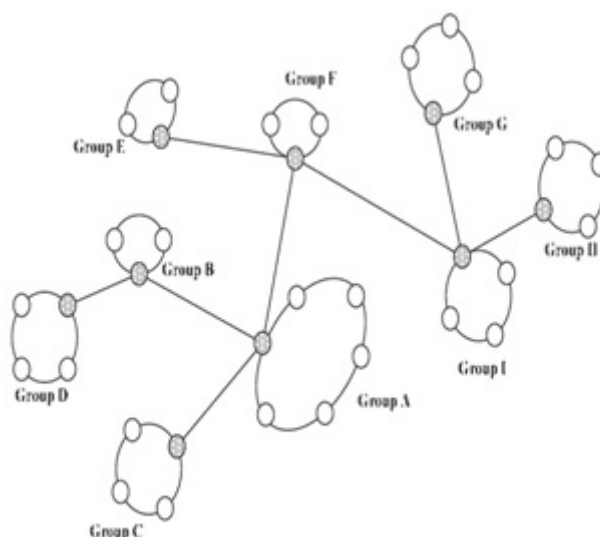


Figure 3: An overview of Malugo system architecture

In particular, all peers are sorted circularly by the alphabetic order of their logical domain names, so the peers belonging to the same logical domain are deployed together, and peers in a logical domain are organized into a ring by an order of numeric identifiers. Searching operation for a file in Boundary Chord system returns the identity of the peer containing the file. First, the client submits the message with the id of the file to an arbitrary peer that checks whether it has the same logical domain name of the id. If so, the peer then determines which peers identifier is immediately following the identifier of required id within its logical domain by sending message to peers in its finger table. Otherwise, the peer sends the searching message to one of the boundary peers in its domain. The boundary max is chosen if the domain name is alphabetically larger than current peers domain name, otherwise, the boundary min gets the searching message. The process of searching continues until the peer containing the required id is found.

Stoica, Morris, Karger, Kaashoek, and Balakrishnan (2001) proposed an efficient lookup algorithm in peer-to-peer networks. Based on Chord structure, given a key, the proposed algorithm maps the key onto the identifier of the node holding the file. Chord provides fast distributed computation of a hash function mapping keys to nodes responsible for the stored data items. With high probability, the hash function balances load. When an Nth node joins/leaves the network, only an $O(1/N)$ fraction of the keys are updated. In an N-node network, each node maintains information about $O(\log N)$ other nodes, and each lookup operation requires $O(\log N)$ messages. The hash function assigns each node and

key an m-bit identifier by hashing the nodes IP address and the key, respectively.

Key K is assigned to the first node whose identifier is equal to or follows K in the identifier space.

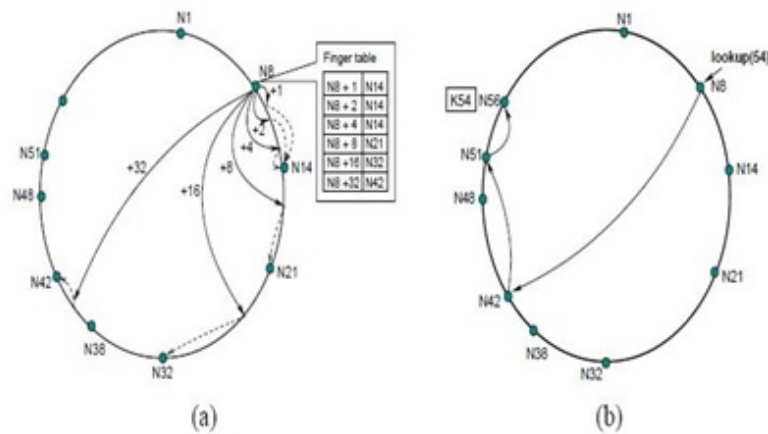


Figure 4: (a) The finger table entries for node 8. (b) The path a query for key 54 starting at node 8, using the algorithm in Figure 5.

2.3 Unstructured Systems

Gkantsidis, Mihail, and Saberi (2005) presented schemes combined flooding and random walks. Flooding is the common search technique in unstructured peer-to-peer networks. The authors considered a hybrid search scheme, called simulation random walk, which can be viewed as a random walk of substantially shorter length combined with shallow flooding on every step of the random walk. The performance of the proposed simulation random walk appears to be better than flooding in regular topologies. This paper gives analytic justification that the simulation of a short random walk with shallow flooding performs well, and rectifies the performance of flooding in the case of a sparse network.

Morselli, Bhattacharjee, Marsh, and Srinivasan (2007) designed an efficient lookup scheme for large peer-to-peer networks. They presented the Local Minima Search (LMS) protocol that uses a virtual name space without imposing specific topologies. It is an efficient alternative for applications in which the topology cannot be structured as a Distributed Hash Table (DHT). LMS introduces the notion of a local minimum; a node u is a local minimum for a data item if and only if the id of u is the closest to the items id in its neighborhood. In general, for any object there are many local minima and replicas are placed onto a subset of these nodes. During a search, random walks are used to locate minima for the given data item.

Sarshar, Roychowdury, and Boykin (2003) proposed an efficient search algorithm for Gnutella peer-to-peer network. The proposed algorithm works as follows. Each node in the network caches its content through a random walk of certain length starting from itself; the contents are duplicated on all visited nodes. A query request is implanted through a random walk of certain length starting from the requester. When the search begins, each node having the query implantation starts a broadcast search, however it only sends a query to a neighbor with prescribed probability.

3 Our Algorithm

In this section, we present an efficient distributed algorithm for unstructured peer-to-peer systems. First we present an algorithm that consists of three levels: storage, bookmarks, and search. Then we introduce a well known set of performance metrics based on them the efficiency of the proposed algorithm can be evaluated.

3.1 Storage Algorithm

Suppose that an arbitrary node u wants to save some file in the system. Node u calculates its degree du (the cardinality of the set $N(u)$ of its neighbors) and asks each of its neighbors for its degree. Then, the node prepares a message containing a copy of the file to be stored in the system, initializes two counters $counter1 = \log n$ and $counter2 = n + 1$. Afterwards, the node chooses a set $Nl(u)$ of the $\lfloor \log(du) \rfloor$ largest degree nodes in the set $N(u)$ and

a random set $Nr(u)$ of cardinality $\lfloor \log(du) \rfloor$ from the set $N(u) - Nl(u)$ of neighbors of u (we choose $Nr(u)$ by allowing every node to choose a random number from 0 to 1: if this number is smaller than $\lfloor \log(du) \rfloor / du$, it will be chosen, and will be ignored otherwise). Node u then sends the message to each node in the set $Nl(u) \cup Nr(u)$. Each node v that receives the message checks the values of $counter1$ and $counter2$. If $counter1$ is positive, it reduces $counter1$ by one, chooses two sets $Nl(v)$ and $Nr(v)$ in the same way as u , and then sends the message to each node in the set $Nl(v) \cup Nr(v)$. If $counter1$ equals zero and $counter2$ equals $n + 1$, then node v updates the value of $counter2$ to be chosen randomly from the space $1, 2, \dots, n$, chooses one of its neighbors randomly to send the message. If $counter1$ equal zero and $counter2$ lies between zero and $n + 1$, then node v reduces $counter2$ by one, and chooses one of its neighbors randomly to send the message. Finally, if both $counter1$ and $counter2$ equal zeros, then node v stores a copy of the file.

Now, we include a set of at most $\log n$ bookmarks for each node that has the file. Every node v which has a copy of the file prepares a message that contains a bookmark and counter with initial value $\log n$, and then sends this message to $\lfloor \log n \rfloor$ nodes from their neighbors randomly. Every node w that receives the message executes one of the following actions. If the counter is greater than zero then node w reduces the counter by one, and then chooses one of its neighbors randomly to send the message. Otherwise (the value of the counter is zero), node w keep bookmarks for v , that is, it has the information that v has the file and keep the path to v . The formal description of the algorithm.

Algorithm Storage

Assume that an initiator node u and the file fu to be stored. Let du denote the degree of u . Node u prepares a message containing fu , and then sends this message to a set $Nl(u)$ of the largest degrees of cardinality $\lfloor \log(du) \rfloor$ in $N(u)$ and to a random set $Nr(u)$ of cardinality $\lfloor \log(du) \rfloor$ in $N(u) - Nl(u)$. Each node v that receives the message proceeds as follows.

1. Node v asks all its neighbors their degrees.
2. If $counter1 > 0$, then v reduces $counter1$ by one, and then sends the message to a set $Nl(v)$ of the largest $\lfloor \log(dv) \rfloor$ degrees in $N(v)$ and to a random set $Nr(v)$ of cardinality $\lfloor \log(dv) \rfloor$ from $N(v) - Nl(v)$.
3. If $counter1 = 0$ and $counter2 = n + 1$, then node v updates the value of $counter2$ with a random number chosen from the space $1; 2; \dots; n$, and then chooses a random node of its neighbors to send the message
1. (we choose the node by make every node chooses a random number from 0 to 1 if this number is smaller than $(1=(dv (\lfloor \log(du) \rfloor)))$).
4. If $counter1 = 0$ and $0 < counter2 < n + 1$, then node v reduces
 2. $counter2$ by one, and then sends the message to a random node of its neighbors.
5. If $counter1 = counter2 = 0$, then node v keeps a copy of the file fu .
6. For each node v that keeps a copy of the file, we apply the following procedure:
 - a. Node v prepares a message containing bookmarks for v with counter
 - b. of initial value $\lfloor \log(n) \rfloor$. This message keeps track of the path from the initiator v .
 - c. Each node w that receives the message will apply one of the follow-ing two actions. If the counter is positive, then node w reduces the counter by one, and then sends the message to a random node of its neighbors. Otherwise (the counter equals zero), node w keeps a bookmark for v . As mentioned before, this

bookmark includes the path to v.

3.2 Search Algorithm

Suppose that node u looks for an arbitrary file in the system. Firstly, it node prepares a message containing the identifier id of the file and a counter initialized by $\log n$. This counter represents the maximum number of nodes to be visited. Then, the node sends the message to $\log n$ randomly chosen neighbors (or the all neighbors of u if the degree of it is at most $\log n$).

Every node v that receives the message will implement one of the following actions. If v has a copy of the file, then it sends it back to u (The message keeps track the path to the initiator node u). If v has a bookmark to the file, then the message will follow the path in the bookmark to get the file and send it back to u. Otherwise (v neither has a copy nor a bookmark of the file), it reduces the counter by one and sends the message to a random neighbor if the counter is positive and ignores the message otherwise. The formal description of the algorithm.

Algorithm Search

Assume that an initiator node u looks for a file f.

1. Node u prepares a message containing the id of the file f, a counter with initial value $\log n$. The message keeps track of the path to the initiator.
2. Node u sends the message to the minimum value of $\{\lfloor \log(n) \rfloor ; d_u\}$ nodes from its neighbors (chosen randomly).
3. Every node v that receives the message executes one of the following actions:
 - If node v has a copy of the file, it sends it back to u using the path in the message.
 - Else if node v has a bookmark to the file, it uses the path in the bookmark to get the file and sends it back to u.
 - Else if the counter is positive, then node v reduces the counter by one, and then sends the message to a random neighbor.
 - Else (the counter equals zero), node v ignores the message (do nothing).

3.3 Performance Metrics

In this section, we present common metrics for evaluating the performance of storage and search algorithms in peer-to-peer networks (Gkantsidis, Mihail, & Saberi, 2005). We will show some of them in details.

Median and Mean: A median is the middle number of a set of sorting number. If the cardinality of the set is even, then the median is the average of the two middle numbers. Mean mathematically equals the sum of a set of numbers divided by the cardinality of the set. In distributed systems, we need to calculate mean and median of distinct peers discovered. Good search algorithms maximize the median and the mean number of distinct peers. The median is a more robust metric especially for topologies with large irregularities in the degrees since it is possible to measure relatively large mean values because few searches may reach a very large number of users and increase the mean value.

Minimum, Maximum, and Standard Deviation of the number

of hits: A large minimum value is important in order to guarantee that the algorithm will have a good worst case performance. The range between the minimum and the maximum values relates to the variation of the performance of the algorithm. The variation is measured using the standard deviation.

Experiment No.	No. of Copies	Start Point	No. of Hops
1	3	19	2
2	4	4	1
3	4	20	1
4	3	4	2
5	3	10	2
6	3	14	2
7	4	8	1
8	2	6	1
9	3	11	2
10	4	16	1

Table 1: Results for $n = 20$

We believe that algorithms with larger minimum values and smaller variation are preferable.

Number of messages: Good search schemes minimize the number of messages used to discover as much information as possible. In order to perform a fair comparison of the different search algorithms we require that they use the same number of messages. Since it is difficult to configure the parameters of each algorithm to guarantee the exact same number of messages, we require that the expected number of messages used in each experiment is approximately the same for all algorithms.

Response time: We also measure the maximum running time of each algorithm. In this study we assume a very simple discrete time model. Each node receives queries from its neighbors and at the same time processes them and forwards copies of the queries, if necessary, to its neighbors. The latter queries will be received at the next unit of time. For all our schemes, it is easy to compute the running time of the algorithm, or an upper bound of it. We believe that our definition of running time can be used to judge the relative performance of the different algorithms.

4 Experimental Results

In this section, we present experimental results obtained by applying our algorithm to random networks of different sizes. We use mean, median, and range to evaluate the number of hops visiting by the algorithm searching for a data item.

Table 5 summarizes the performance metrics of the results shown in tables

ExperimentNo.	No. of Copies	Start Point	No. of Hops
1	3	19	2
2	4	4	1
3	4	20	1
4	3	4	2
5	3	10	2
6	3	14	2
7	4	8	1
8	2	6	1
9	3	11	2
10	4	16	1

Table 1: Results for $n = 20$

ExperimentNo.	No. of Copies	Start point	No. of Hops
1	6	44	1
2	6	21	1
3	5	1	2
4	6	32	1
5	6	22	1
6	8	26	1
7	7	47	1
8	8	19	1
9	5	13	2
10	6	50	1

Table 2: Results for $n = 50$

ExperimentNo.	No. of Copies	Start Point	No. of Hops
1	10	10	1
2	10	30	1
3	10	90	1
4	10	100	3
5	10	1	2
6	10	54	1
7	10	21	1
8	10	17	1
9	10	41	1
10	10	69	3

Table 3: Results for $n = 100$

Experiment No.	No. of Copies	Start Point	No. of Hops
1	16	900	1
2	16	55	1
3	16	666	4
4	16	5	1
5	16	333	5
6	16	777	1
7	16	25	1
8	16	145	1
9	16	3	1
10	16	15	1

Table 4: Results for $n = 1000$

n	Mean	Median	Range	Average No. of Copies
20	1.5	1.5	1:2	3.3
50	1.2	1	1:2	6.3
100	1.5	1	1:3	10
1000	1.7	1	1:5	16

Table 5: Performance metrics of the results in Tables 1-4

5 Conclusion

It is well known that peer-to-peer storage systems are classified into structured and unstructured systems. Both structured and unstructured systems try to meet two conflict goals: increasing the success rate within reasonable response time and decreasing the worst case message complexity. We have focused on designing an efficient algorithm for unstructured peer-to-peer networks. We have proposed an algorithm for each of the main operations in storage systems: storage and search. In the storage, we have presented an algorithm for storing files in which we have maintained a reasonable number of copies of each file such that these copies can be reached in almost the same time from all nodes. In the search algorithm, we have used a reasonable number of messages from the node that requests the file such that they cover random parts of the system. Experimental results show that the proposed algorithms have good success rate and response time with reasonable number of messages. As a future work, we plan to extend the proposed algorithm to dynamic systems in which nodes may leave and other nodes may join the system.

References

- Chan, Y. W., Ho, T. H., Shih, P. C., & Chung, Y. C. (2010). Malugo: A peer-to-peer storage system. *International Journal of Ad Hoc and Ubiquitous Computing*, 5(4), 209-218.
- Gkantsidis, C., Mihail, M., & Saberi, A. (2005). Hybrid search schemes for unstructured peer-to-peer networks. In *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2005)*, 1526-1537.
- Jin, H., Wang, C., & Chen, H. (2005). Boundary Chord: a novel peer-to-peer algorithm for replica location mechanism in grid environment. In *Proceedings of the 8th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN 2005)*, 262-267.
- Morselli, R., Bhattacharjee, B., Marsh, M. A., & Srinivasan, A. (2007). Efficient lookup on

unstructured topologies. IEEE Journal on selected areas in communications, 25(1), 62-72.

Sarshar, N. I. M. A., Roychowdury, V., & Boykin, P. O. (2003). Percolation-based search on unstructured peer-to-peer networks. In Proceedings of the 2nd International Workshop on Peer-to-Peer Systems, LNCS 2735, 2-9.

Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., & Balakrishnan, H. (2003). Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Transactions on Networking, 11(1), 17-32.

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., & Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. ACM SIGCOMM Computer Communication Review, 31(4), 149-160.

Tanenbaum, A. S. (2011). Computer Networks, /Andrew S. Tanenbaum, David J. Wetherall. Cloth: Prentice Hall.

Tanenbaum, A. S., & Van Steen, M. (2007). Distributed systems: principles and paradigms. Prentice-Hall.



© 2016 The Author(s). This open access article is distributed under a Creative Commons Attribution (CC-BY) 4.0 license.

You are free to: Share — copy and redistribute the material in any medium or format. Adapt - remix, transform, and build upon the material for any non-commercial purpose. The licensor cannot revoke these freedoms as long as you follow the license terms. Under the following terms: Attribution -You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use. No additional restrictions. You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits