Peer-to-Peer Netw. Appl. (2017) 10:208–215
DOI 10.1007/s12083-015-0420-5

Recommend top-k most downloaded files in the chord-based P2P file-sharing system

Sina Keshvadi[1] · **Amir Masoud Rahmani**[2] · **Habib Rostami**[3]

Received: 5 February 2015 / Accepted: 26 October 2015 / Published online: 10 November 2015

Abstract Peer-to-peer (P2P) file sharing system provides a platform that enables users to share their files over network and provides a rapid and scalable content distribution mechanism to access an explosive volume of shared files on the system. The search mechanism in these systems is mainly keyword-based, so, users may have to try different keywords to find the desired file, where increase the network traffic. Therefore, the existence of a tool in these clients that guide users towards the top-k most downloaded files is very beneficial in terms of user convenience and traffic reduction. We have proposed our algorithms by adding a new data structure to chord protocol. Chord has been widely used as a routing protocol in structured P2P networks. We validated our proposed algorithms through simulation by using PlanetSim simulator and studied the effect of several parameters on the performance of our algorithms. The results show very good performance, in terms of communication cost and response time.

Keywords DHT · Chord protocol · File sharing system · Top-k queries

✉ Sina Keshvadi
Keshvadi@pnu.ac.ir

Amir Masoud Rahmani
Rahmani@srbiau.ac.ir

Habib Rostami
Habib@pgu.ac.ir

[1] Department of IT, Payame Noor University (PNU), Tehran, Iran

[2] Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran

[3] Computer Engineering Department, School of Engineering, Persian Gulf University, Bushehr, Iran

1 Introduction

Peer-to-Peer (P2P) networks came to enable users to share limited resources in exchange with others and have access to many others resources. P2P systems can be characterized as distributed systems in which all the peers offering identical responsibility, decentralized control, scalability, autonomy and self-configuration [1, 2]. P2P networks are good platforms for large-scale file sharing systems and is interested in both academic and industry communities [3–5] (e.g. Napster, Gnutella, Kazaa and BitTorrent). In these systems, user connects to the network via P2P client, shares his/her files on system and downloads desired files. To retrieve a file, client generates a routing request, sends to other peers according to the underlying routing algorithm and then receive the location information of the requested file and downloads it finally. The search mechanism in these systems is mainly keyword-based [6], so, users may have to try different keywords to find the desired file, where increase the network traffic. Therefore, the existence of a tool in these clients that guide users towards the top-k most downloaded files is very beneficial in terms of user convenience and traffic reduction [7].

Some P2P networks use Distributed Hash Tables (DHTs) to store and retrieve files on the network. Distributed Hash Tables (DHTs), e.g. CAN, Chord, Tapestry and Pastry, provide an efficient solution for data location and lookup in large-scale P2P systems. They all map a given key onto a peer *p* using a hash function and can lookup *p* efficiently; usually in O (log n) routing hops where n is the number of peers. In order to suggest the k most downloaded files, we should determine the total number of downloads of all copies of a file in the system and such queries are referred as top-k queries. The formal definition of a top-k query is as follows [8]:

Let *Net* be a P2P file-sharing network with N nodes. Let f_{ij} is a copy of file i hosted by node j and $dl(f_{ij})$ is number of downloads of file i form node j. Also, we define $dl(f_i) = \sum_{j\ is\ a\ peer\ in\ Net} dl(f\,ij)$. It means that $dl(f_i)$ is the total number of downloads of various copies of file f_i from different peers in the *Net*. Each node in the *Net* keeps a list of the files it hosting, which is sorted by their number of downloads. We define score of file i as follows [8]:

$$\mathrm{Score}(fi) = dl(fi) = \sum_{j=1}^{N} dl\,\mathrm{f}_{ij} \tag{1}$$

We use the score function to rank the files. In the other words, whenever a node requests the top-k list, in response to the top-k query, a list includes k files names with the highest scores is returned.

The rest of this paper is organized as follows. In part 2, we explain related work in this area. In the part 3, we provide an overview on chord protocol and explain our proposed data storage mechanism on it. We present our proposed algorithms in part 4. Next, in part 5, we report the performance evaluation of our algorithms through simulation. Finally, we make a brief concluding remark in part 6.

2 Related work

To the best of our knowledge, there is no efficient work that gets Top-k most downloaded list in less than three phases over large and dynamic file sharing systems. Many works are on centralized or Client-Server systems that a complete survey of these algorithms said in [9]. A naive distributed algorithm is Fagin's Algorithm (FA) [10] that gets all complete scores lists of all files from all peers, compute the overall scores of any file, and return the k top most downloaded files. This algorithm is executed in O(m*n) (m files and n peers) and thus uses a large number unnecessary accesses and it is inefficient for wide-size P2P file sharing systems. The main algorithm proposed so far to answering top-k queries over sorted lists is the Threshold Algorithm (TA) [11]. The basic difference between TA and FA is its stopping process when the overall score is greater than a threshold.

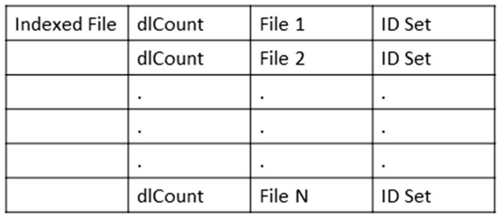
TA goes down the sorted lists in parallel, one position at a time, and for each seen object, computes its total score. This process continues until finding k objects whose overall scores are greater than a threshold, which computed based on the local scores of the objects at current position. TA needs huge amount of interaction and communication and in worse case, TA needs same cost with FA [11].

The TPUT (Three-Phase Uniform Threshold) [12] uses three phases in order to resolve top-k queries in star topologies and it is the first fixed-round algorithm designed to single-hop networks. The disadvantage of TPUT is that the threshold ***s*** is uniform for all nodes, where it results in the unnecessary transfer of many objects that are not in the final Top-k result. In [13] they propose the Threshold Join Algorithm (TJA), use TPUT's idea in TA algorithm that utilizes a non-uniform threshold on the attribute in order to minimize the transfer of data. [14] propose a TA-Based algorithm to processing top-k queries over sorted lists. They propose the best position algorithm (BPA), which the key idea of BPA is that its stopping mechanism takes into account special seen positions in the lists, the best positions. In [15], they proposed a location-aware system that returns ranked objects that are near a query location and have textual descriptions that match query keywords. They proposed the concept of prestige-based relevance to capture both the textual relevance of an object to a query and the effects of nearby objects. A review of the spatial keyword querying functionality is in [16]. The spatial keyword querying takes a user location and user-supplied keywords as arguments and returns top web objects that are spatially and textually relevant to these arguments.

3 An overview on chord protocol and propose data storage mechanism

In this section, we first describe the Chord protocol and then we propose an efficient data storage mechanism based on this protocol. Chord [17] is a distributed look up protocol that provides a completely decentralized mechanism where allows a node to look up an object in O(log N) by tracking only O(log N) locations and constructs a logical ring within all N participating nodes. It supports just one operation: given a key, it maps the key onto a node responsible for them with consistent hashing function, like SHA-1 [18], which has several desirable properties to assign an m-bit identifier to each node and each key. A node's identifier chosen by hashing the node's IP address, while key identifier produced by hashing the key. Look up done with O(log N) messages, while join and leave operations take no more than O(log N) with high probability. For a more detailed introduction to Chord, please refer to [17] and how consist hashing function work refer to [18].

Each node in the network keeps two lists of objects. The first list contains the name and other conventional attributes of a file that the node has shared as well as its number of downloads (dl). We call this list the *OwnFile* structure. The second list, the *indexed file* structure, indexes the files assigned to the node based on the chord protocol over DHT. The proposed structure is shown in the Fig. 1.



Indexed File	dlCount	File 1	ID Set
	dlCount	File 2	ID Set
	.	.	.
	.	.	.
	.	.	.
	dlCount	File N	ID Set

Fig. 1 Indexed file structure

In this structure, the ID Set of file F_i is the set of addresses of the nodes that have kept a copy of file F_i. If several copies of file F_i are available in the network and according to Chord's hash function the node n is responsible for storing F_i, then the node n stores F_i and its ID Set, which contains the addresses of all of the nodes that have shared F_i. Also, the node n stores the total number of downloads of the file F_i in addition to the ID set. Therefore, the system will no longer be in a state where the same copies of a file are distributed across the system. In order to determine the number of downloads of a file F_i, we just need to find its responsible node (the node which keeps the addresses of the nodes hosting a copy of F_i) using the basic chord lookup operation.

For example in Fig. 2, we have used our proposed data structure to store a sample network with 4 nodes and 8 files based on chord protocol.

4 Proposed algorithms to list the top-k most downloaded files

In this section, we describe our proposed algorithms by using data storage mechanism that mentioned in pervious part to perform top-k query in the P2P file sharing system.

4.1 DHTTop algorithm

First, we must determine how a peer downloads a desired file over the proposed structure. The download process is as follow:

```
// peer n wants to download file f
key = n.Hash(f.filename);
N_r = n.Find (key);
N_o = N_r.getOwner (key);
n.request (N_o, key);
if (n.download(key))
{
        N_r.Increase (key , 1);
        n.addOwnFile (f);
}
```

As soon as the node n decides to download file f, it hashes the filename of f using a hash function on DHT and gets the key. Then the DHT looks up the node N_R, which is responsible for addressing the key. N_R returns the address of one of the nodes sharing f in the system (which we call N_O) to node n. Now, the node n requests the f from N_O. After downloading the f, it notifies N_R to increase the dlcount of f by one unit.

When a node p requests the top-k list, it sends a broadcast message to all its neighbors. We use the broadcasting algorithm provided in [19], which presents an efficient mechanism for performing a broadcast operation with minimal cost in log (n) steps and exactly n-1 messages, where n is the number of peers.

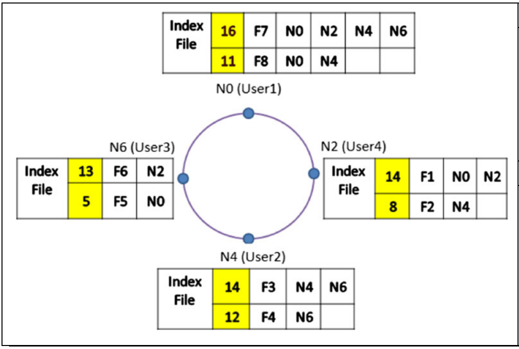
After sending a request, in the reverse path of broadcasting, each peer receives a message from its children containing their top-k list. Then the peer appends its own local top-k files to the message and sends it to its parent node, all the way back to the requesting peer (p). Finally, p performs the selection sort on all of the received items and finds the k files with maximum dlcounts.

4.2 RPDHT algorithm

In this section, we modify our naive algorithm, DHTTop, by improving its performance in the reverse path of the broadcast. When a peer receives the top-k lists form its children, it uses selection sort to combine the received lists with its own local top-k list, and only sends the best k files to its parent. At the end, the root receives the final top-k list from its children. On the reverse path of top-k request, the message size is greatly reduced, because the size of each message is fixed and equal to k filenames. Therefore, using this method leads to a significant reduction in the network traffic and thereby reduces the response time too.

4.3 Δt-based DHT algorithm

Top-k algorithms from one perspective can be divided into two categories: exact answers and approximate answers. In this section, we propose an approximation algorithm with good accuracy. As soon as a peer receives the top-k list, we



Peer1		dl	Peer3		dl
Hosted File	F_1	7	Hosted File	F_4	12
	F_8	5		F_3	8
	F_5	5		F_7	7
	F_7	4			
Peer2		dl	Peer4		dl
Hosted File	F_6	13	Hosted File	F_2	8
	F_1	7		F_8	6
	F_7	4		F_3	6
				F_7	4

Fig. 2 Implement a simple file sharing system on chord protocol

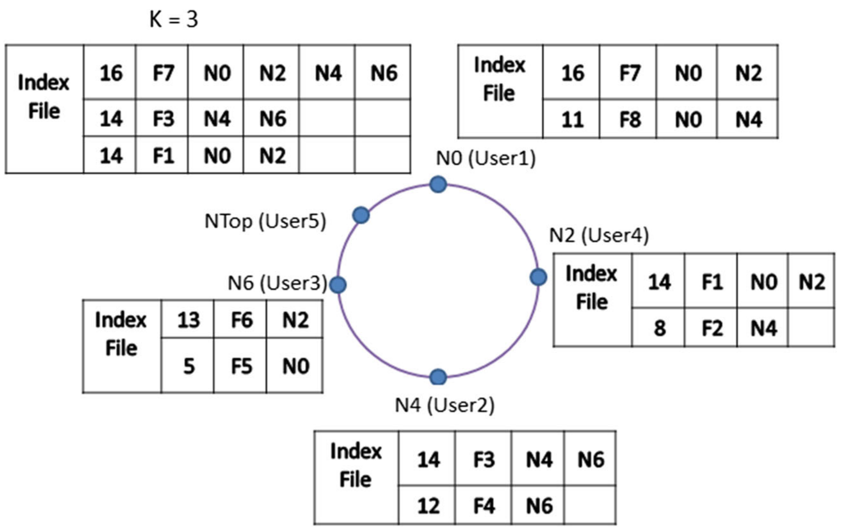


Fig. 3 Implement Fig. 2 over chord protocol by DHTNodeTop technique

assign an expiration time ΔT to the list, which indicates that the list is valid for the time interval ΔT. When a peer broadcasts its top-k request, if a peer in the broadcast path has a top-k list that has not expired yet, instead of sending requests to its neighbors, it returns the existing top-k list and its expiration time ΔT to its parent. In addition, if a peer receives several top-k lists with different ΔTs, it chooses the most recent one and if the received list is newer, sets it as its own local top-k list and returns it to its parent. We have studied the results of this approach in Section 5.6.

4.4 DHTTopNode algorithm

We use a super peer called TopNode to continuously keep the top-k list in the file sharing system. For our goals, TopNode must be constructed form peers that are relatively powerful and stable. The power of a peer can measure by its bandwidth, memory and CPU speed. Stability can measure by the average length of continuous time that a peer is available. The proposed mechanism to file downloading in DHTTopNode algorithm is as follows:

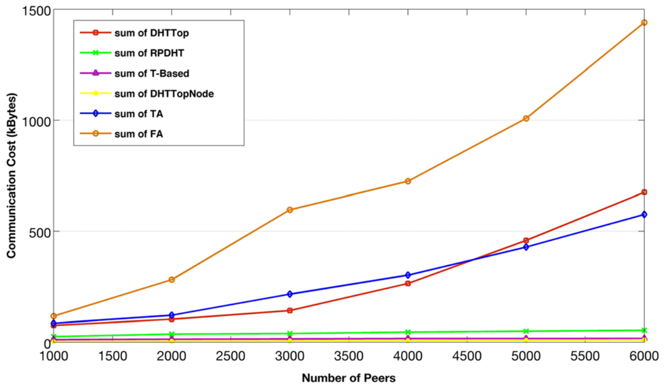
```
// n wants downloads f
key = n.Hash (f.filename);
N_r = n.Find (key);
N_o = N_r.getOwner (key);
n.request (N_o, key);
If (n.download (key) == true)
    N_r.Increase (key, 1);
    n.addOwnFile (f);
    n.SendMessage(TopNode, f, dlCount);
```

```
//Top-node process after receive a download message
if(top-kList.contain(f))
  top-kList.Replace(f, f, dlCount);
else
   if (top-kList.Length() < k)
          top-kList.append(f, dlCount);
          top-kList.sort();
   else If(top-kList[k].dlCount<f.dlCount)
        top-kList.Replace(top-kList[k], f, dlCount);
```

When peer *n* looks up a file *f* in the system and downloads it, N_R (the node responsible for indexing *f* in the chord) increases the dlcount of file *f* and sends a message containing the name of file *f* and its new dlcount value to TopNode. When

Table 1 Default settings for experimental parameters

Parameter	Default values	Parameter	Default values
Number of Shared Files	3500	Download Time	Between 10 and 100 s
Default *K*	10	Query arrival rate	50 queries per seconds from 1000 peer
Average Own File for each Peer	20	Default Δt	40 s
Downloaded File	Between 20 and 30	Bandwidth dataset	King data set http://pdos.csail.mit.edu/p2psim/kingdata
User and their Files dataset	Movie Lens	Latency dataset	MIT data set
Average Filename Size	30 byte	Default Network Size	1000 Peer



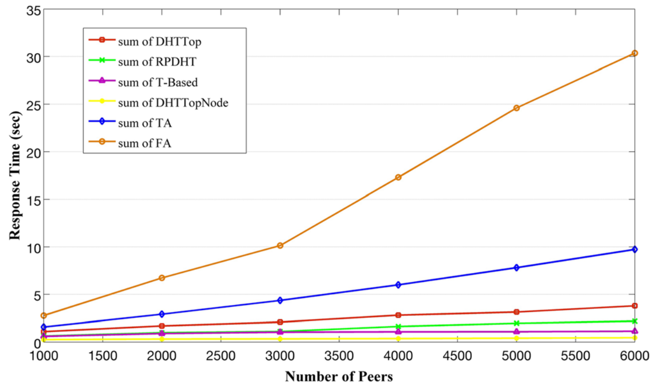


Fig. 4 Effect of scalability on communication cost and response time

TopNode receives the message, it searches the top-kList for *f* and if it is found, replaces the old value of *f*'s dlcount with the new one and sorts the top-kList. If the top-kList does not contain *f*, it checks the size of top-kList and if it is lower than k, adds *f* to the end of the top-kList. Otherwise, TopNode compares the dlcount of *f* with the last dlcount in the top-kList. If it is greater, it replaces the kth item in the top-kList with *f* and sorts the top-kList using a bottom-up algorithm.

In the proposed structure, when a peer requests the top-k list or a new nodes joins the system, it can get the top-k list from the TopNode directly. We have stored the network of Fig. 2 in Fig. 3, using this mechanism.

In next section, we evaluate our algorithms through simulation.

5 Performance evaluation and simulation setup

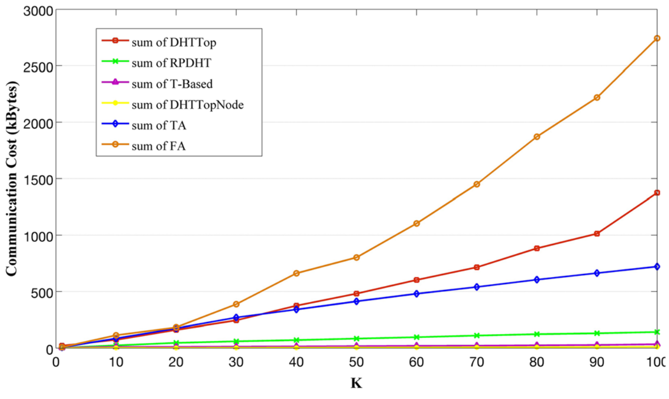
We implemented and evaluated our algorithms using by PlanetSim simulator [20]. PlanetSim is an open source and java based framework that implement and evaluate overlay networks such as Chord and Symphony. Our default settings and network parameters are shown in Table 1.

To evaluate the performance of our algorithms, we measure the following metrics. 1) Response time, the time elapsed between the delivery of top-k request to DHT and the show top-k list; 2) Communication cost, the total number of bytes, where transferred over the network for executing algorithms, look-up and sorting Top-k list.

In order evaluate our works, we simulate, tested and compared two most used algorithms i.e. FA and TA with four versions of our algorithms.

5.1 Scalability

We investigate the scalability on response time and communication cost of our algorithms. As we seen in Fig. 4, DHTopNode has minimum response time and communication cost against others. Next, RPDHT and DHTTop algorithms are better than TA where return top-k list in less time and cost toward TA. It must be noted that the length of the messages gets larger and larger with each hop of the reverse path of DHTTop. In the Δt-based method, increasing the network size will increase the chance of receiving fresh list from neighbors. Hence, increasing the number of nodes improves the performance. Finally, basic algorithm i.e. Fagin-based method is worse time rather than other algorithms.



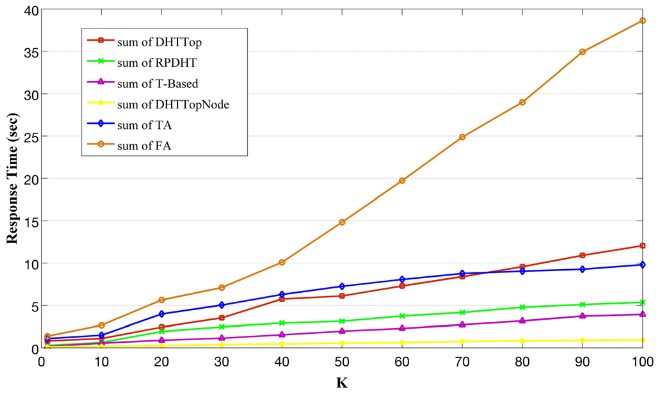


Fig. 5 Effect of K on communication cost and response time

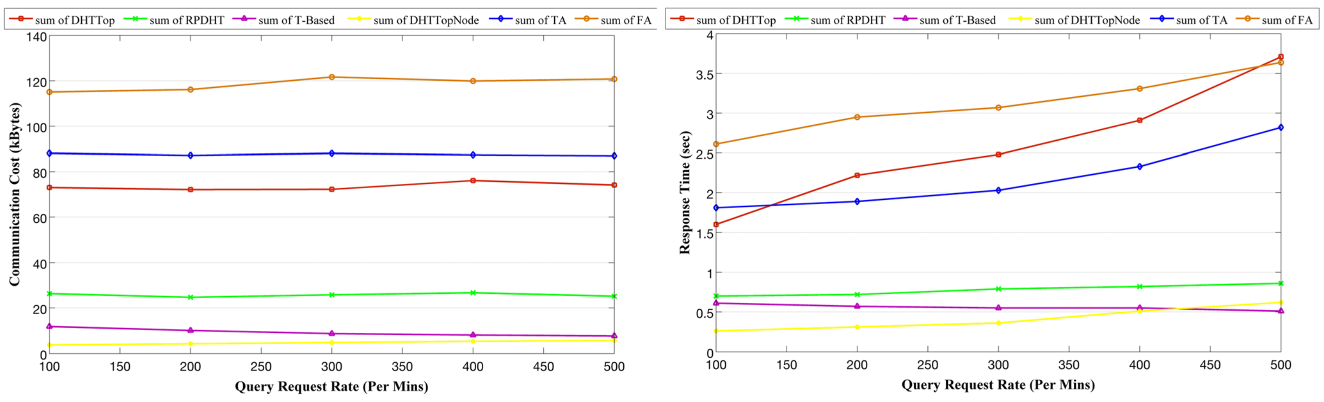


Fig. 6 Effect request rate on communication cost and response time

5.2 The effect of k

As we seen in Fig. 5, by increasing k, TA and FA is required to more additional transition to find the top-k list. In addition, when the k is large, e.g. more than 100, the performance of the DHTTop greatly reduced against the RPDHT. As we can see, in the DHTTopNode method the response time and communication cost remains uniform while increasing k, because it gets the top-k list directly from TopNode. The performance of Δt-based algorithm is better than RPDHT but by increasing k, its accuracy may be reduced which we have studied this issue in Section 5.5.

5.3 Effect of arrival request rate

We have increased the arrival rate of top-k requests from 100 to 500 per minute in a network of 1000 peers. As it can be seen in Fig. 6, the growth rate of network traffic and response time of requests are uniform in both RPDHT and DHTTop algorithms. This is also true about TA and FA. However, this change has quite different effects on the ΔT-based algorithm. With the increase of requests rate, the probability of having a recent top-k list also increases and the time and traffic cost of each request decreases. In the DHTTopNode algorithm, the volume of messages and the number of input/output messages of the TopNode and its adjacent neighbors increase with the increase of requests rate, so TopNode may suffer from bottleneck. In this case, we can use some replicas across networks to balance the load.

5.4 The effect of number of shared files

As mentioned before, each peer in the system has a folder for keeping and sharing its OwnFiles in the system. In addition, most peers share the downloaded file over the system. So, the same copies of a file is distributed across the system. Assuming that the variety of files in the system is constant, we have changed the number of shared files and studied its effect on the network.

As we can see in Fig. 7, increasing the number of shared files has little effect on the performance of our main algorithms. In the TA, by increasing the number of shared files, the files will be distributed across more nodes in the system and therefore, more passes and time are required to find each element of Top-k list.

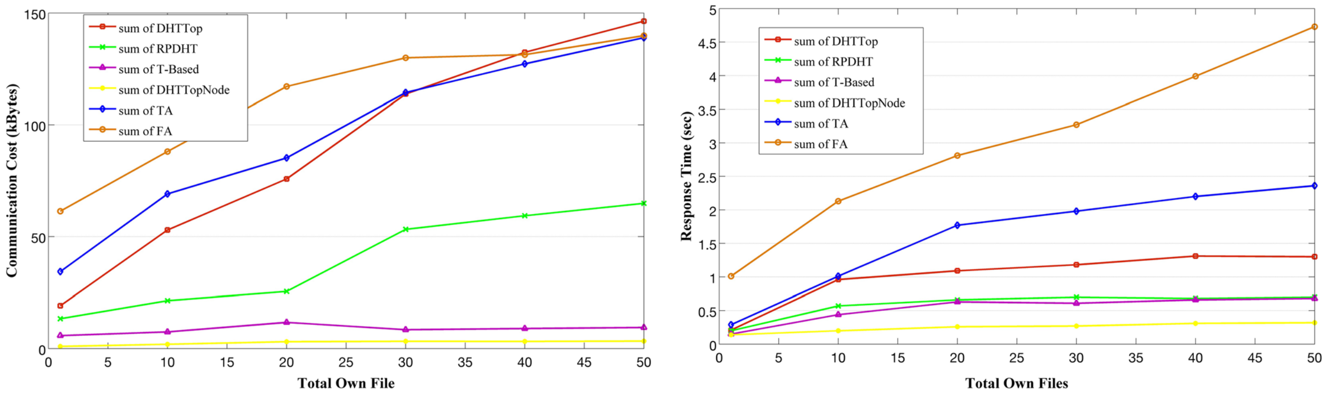


Fig. 7 Effect of number of shared files on communication cost and response time

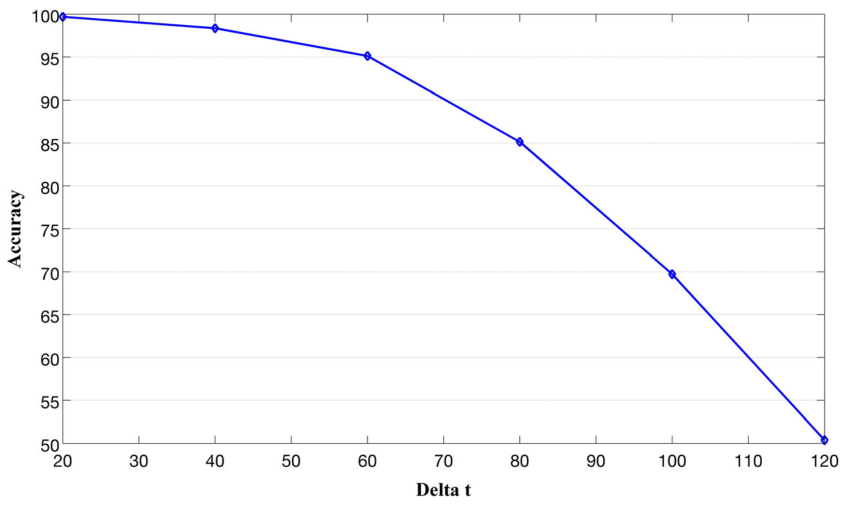


Fig. 8 Accuracy of ΔT-based algorithm

5.5 Accuracy of the Δt-based algorithm

Here, we want to study the accuracy of the Δt-based algorithm in difference *ΔT*. First, we should define the accuracy of the results. Given a top-k request Q, let U be the exact set of k top results owned by the peers that received Q, U' be the set of top-k results which are returned to the requested peer via Δt-based Algorithm. We denote the accuracy of results by ac_Q, and define it as:

$$ac_Q = \frac{\left| U \cap U' \right|}{|U|} \quad (2)$$

We use default setting in the network as mentioned in Table 1. As is seen in Fig. 8, accuracy of the top-k for small *ΔT* i.e. 20 s is 99.7 %. For longer*ΔTs*, accuracy decrease because downloaded times of files and dlCount change continuously in the system.

6 Conclusion and future work

In this paper, we addressed the problem of top-k most popular file in P2P file sharing systems. We first proposed a new data storage mechanism in DHTs, which provides a good support for top-k queries. Then, we proposed four efficient algorithms for ranking top-k most downloaded files in P2P file sharing system. Our naive algorithm, DHTTop, broadcasts the top-k query over chord-protocol and receives top-k list from its children. Then, RPDHT algorithm, modify our naive algorithm by improving its performance in the reverse path of the broadcast. Third, we propose an approximation algorithm with good accuracy named Δt-Based Algorithm. It assigns an expiration time, ΔT, to the each top-k list. Finally, we use a super peer called TopNode to keep continuously the top-k list in the file sharing system. Our algorithms is not TA-style, it is much more general since it supports a large set of any scoring functions. We validated our algorithm through simulation by using PlanetSim simulator and studied the effect of several parameters on the performance of our algorithms. The results show very good performance, in terms of communication cost and response time. As we see, DHTTop algorithm has the best performance rather than others.

We intend to perform other top-k query functions in the DHTs by using our proposed data storage mechanism. We also intend to implement a recommender system to be added to Chord-based P2P file sharing system and towards clients to desired files.

Compliance with ethical standards

Ethical statement "If you do not have integrity, you have nothing. You cannot buy it. You can have all the money in the world, but if you are not a moral or ethical person, you really have nothing." I agree with this view. As a researcher, I abide by some basic personal ethics that help me become a better person everywhere and every day. I have immense respect for my professors, for other researchers, and for readers. Integrity and honesty are two values which I promise to abide by in every situation. I will never engage in plagiarism, cheat, or break any rules, which might result in someone else getting hurt. I promise to stand up against all that is wrong, and will always support nothing but the truth.

References

1. Felber P, Kropf P, Schiller E, Serbu S (2014) Survey on load balancing in peer-to-peer distributed hash tables. IEEE Commun Surv Tutorials 16(1):473–492

2. R. Bolla, R. Gaeta, A. Magnetto, M. Sciuto, M. Sereno. A measurement study supporting P2P file-sharing community models. Comput Netw, vol. 53, Issue 4, pp. 485–500, 2009.
3. Li C, Yu B, Sycara K (2009) An incentive mechanism for message relaying in unstructured peer-to-peer systems. Electron Commer Res Appl 8:315–326
4. Xu Z, He X, Bhuyan L (2006) Efficient file sharing strategy in DHT based P2P systems. Comput Commun 29:1243–1259
5. Zhan Su, Anthony K. H. Tung, Zhenjie Zhang. Supporting top-K item exchange recommendations in large online communities, EDBT 2012, March 26–30, 2012, Berlin, Germany.
6. Rostami H, Habibi J, Livani E (2008) Semantic routing of search queries in p2p networks. J Parallel Distrib Comput 68(12):1590–1602
7. Ekstrand MD, Riedl JT, Konstan JA (2010) Collaborative filtering recommender systems. Foundations and Trends in Human-Computer Interaction 4(2):81–173
8. B. Sanyal, P. Gupta, S. Majumder. Top-K range-aggregate queries on categorical data, in *Emerging Trends and Applications in Computer Science* (*NCETACS*), 2012.
9. Ilyas IF, Beskales G, Soliman MA (2008) A survey of top-k query processing techniques in relational database systems. ACM Comput Surv 40
10. R. Fagin. Combining fuzzy information from multiple systems, in *Proceedings of the 15th ACM Symposium on Principles of Database Systems* (*PODS*), pp. 83–99 1999.
11. Fagin R, Lotem J, Naor M (2003) Optimal aggregation algorithms for middleware. J Comput Syst Sci 66(4):614–656
12. Pei Cao, Zhe Wang. Efficient top-K Query Calculation in Distributed Networks, in *Proceedings of the Twenty-third Annual ACM Symposium on Principles of Distributed Computing*, pp. 206–215, Canada, 2004.
13. Demetrios Zeinalipour-Yazti et. al. Finding the K highest-ranked answers in a distributed network. Comput Netw, vol. 53, pp. 1431–1449, 2009.
14. Akbarinia R, Pacitti E, Valduriez P (2011) Best position algorithms for efficient top-k query processing. Inf Syst 36:973–989
15. Xin Cao, Gao Cong, Christian S. Jensen. Retrieving Top-k Prestige-Based Relevant Spatial Web Objects, Proceedings of the VLDB Endowment, Vol. 3, No. 1,, September 13–17, 2010, Singapore.
16. Xin Cao, Lisi Chen, Gao Cong, Christian S. Jensen, Qiang Qu, Anders Skovsgaard, Dingming Wu, Man Lung Yiu. Spatial keyword querying, LNCS 7532, pp. 16–29, Springer-Verlag Berlin Heidelberg 2012.
17. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-topeer lookup service for internet applications, in *SIGCOMM*, 2001.
18. FIPS 180–1. Secure hash standard, in *Technical report*, *US Department of Commerce*/*NIST*, http://www.itl.nist.gov/fipspubs/fip180-1.htm, April 1995.
19. El-Ansary S, Alima LO, Brand P, Haridi S (2003) Efficient broadcast in structured P2P networks. IPTPS:304–314
20. P. Garcia, et al. Planetsim: A new overlay network simulation framework, in *In. Proc. of ASE*, 2004.

Sina Keshvadi received his M.S. degree in computer science from University of Branch and Research of Ahvaz in Iran, in 2010. He is now faculty member of the Information Technology Department, PNU University of Iran. His current research interests are in the area of distributed computing, P2P systems and cloud computing. Email: Keshvadi@pnu.ac.ir

Amir Masoud Rahmani received his BS in Computer Engineering from Amir Kabir University, Tehran, in 1996, the MS in Computer Engineering from Sharif University of Technology, Tehran, in 1998 and the PhD degree in Computer Engineering from IAU University, Tehran, in 2005. Currently, he is an Associate Professor in the Department of Computer Engineering at the IAU University. He is the author/co-author of more than 120 publications in technical journals and conferences. His research interests are in the areas of distributed systems, ad hoc and wireless sensor networks and evolutionary computing.

Habib Rostami received his B. Sc, M.Sc and Ph.D in computer engineering from sharif university of technology. Now, he is an assistant professor of computer engineering at Persian Gulf University of Bushehr, Iran. His interest includes P2P networks, Data Mining and soft computing.