An Enhanced Data Replication Method in P2P Systems

Somayeh Mohammadi, Hossien Pedram, Somayeh Abdi, and Ali Farrokhian

Abstract— P2P applications emerged as a powerful paradigm for sharing information across the internet. They cause much of internet traffic. The unstructured P2P systems usually use a bounded flooding method for searching. Observations have shown that a few of peers share most of data in a file sharing system. In order to increase the success rate of blind search and load balancing, replication techniques are used in these systems. This paper presents an enhanced data replication method for pure unstructured P2P file sharing systems such as Gnutella, based on popularity of files. Proposed method uses both of index and file replication. Simulation results show that this method has the best performance than some others, although it have a bit more cost.

Index Terms— P2P, Data Replication, Unstructured, File Popularity

u

1 Introduction

P2P systems were classified by [1] into three different categories. Some P2P systems, such as Napster [2], are centralized in that they have a central directory server to which users can submit queries. Other P2P systems are decentralized and have no central server; the hosts form an ad hoc network and send their queries to other peers. Of these decentralized designs, some are structured in that they have close coupling between the P2P network topology and the location of data. Other decentralized P2P systems, such as Gnutella [3] are unstructured with no coupling between topology and data location. In these systems, peers use the bounded flooding mechanism (to search the resource of the network and directly exchange this resource. This model do not lead to the paralysis of the entire network because of the failure of some nodes in the network, but the flooding search mechanism has poor efficiency, and will bring a lot of power exponential growth of the request message number during the process of the resource search.

Data replication consists of maintaining multiple copies of data, called "replicas" on separate computers [4]. Replication is widely used in distributed data management with read-intensive workloads for a number of reasons [5] such as:

1. P2P network is a dynamic self-organized network, in which peer can freely join or leave, so there will lost a lot of important data when some important nodes fail. This feature is not good to the expansion of the P2P application, so replication is used to enhance the reliability of the network.
2. The unstructured P2P systems usually use a blind flooding method for searching. Where a query initiator sends requests to all or a randomly chosen subset of network neighbors and these requests are forwarded up to some pre specified depth or when a preset time-to-live (TTL) expires. Implementation of flooding is simple but it does not guarantee finding the requested data item. In order to increase the success rate of blind search and data availability, replication techniques are needed in these systems.
3. In a P2P system, if there is a large number of download requests and not so many peers are sharing the desired file(s), some peers will become overloaded with download requests that could be satisfied by other peers which host replicas of the requested file(s). By replicating the popular files at more than one peer, access requests have a choice among multiple replicas; load balance can be improved and results in higher throughput and shorter response times.

The rest of the paper is organized as follows: Section 2 presents the related work. In section 3 we propose an enhanced replication strategy, in section 4 we present a simulation model, Section 5 presents the simulation results and comparison of our proposed strategy with other strategies, and finally in section 6 we conclude the paper.

2 Related Work

Replication has been much studied in structured peer-to-peer systems. For instance, Beehive [6] and EpiChord [7] are approaches tailored for DHT routing mechanisms or proposed strategies in [8]. But most of their approaches are not applicable to unstructured peer-to-peer systems.

A widely recognized analysis of replication in unstructured P2P networks is the work of Cohen and Shenker [9]. They investigate the question which replication strategy minimizes the expected number of peers that have to be

- *Somayeh Mohammadi is with Computer Engineering Department, Ghasreshirin Branch Islamic Azad University, Ghasreshirin, Iran*
- *Hossein Pedram is with Department of Computer Engineering and Information technology Amirkabir University of technology, Tehran, Iran*
- *Somayeh Abdi is with Computer Engineering Department, Eslamabadegharb Brach, Islamic Azad University, Eslamabade Gharb, Iran*
- *Ali Farrokhian is with Computer Engineering Department, Razi University, Kermanshah, Iran*

sampled until the requested data items are found (or query size). The main constraint in this setting is the storage capacities of peers. This approach just tries to minimize maximum query size by replicating pointers, instead of balancing load or reducing average download time by replicating files.

Sozio et al. [5] extend the approach of [9] by explicitly addressing the question where to place replicas. They propose a distributed replication algorithm that was tailored to realistic search protocols and achieved near-optimal replica placements with a proven performance guarantee.

Another interesting result is the work of Tsoumakos et al. [10]. Their approach couples lookup indices along with an aging mechanism in order to identify, in real time, query intensive areas inside the unstructured overlay. Peers then individually decide on the time and extent of replication, based on local workload computation. APRE adaptively expands or contracts the replica set of an object in order to improve the sharing process and achieve a low load distribution among the providers. The scheme proves particularly useful in the event of flash crowds, managing to adapt quickly to sudden surges in load process.

In [11] authors proposed a decentralized model for dynamic creation of replicas in a decentralized P2P network. The aim was to ensure a specified degree of data availability.

Paper [11] proposes a distributed replication mechanism for decentralized unstructured P2P networks which consider content popularity. An index of the popular content will be allocated on the node halfway between the requester and the provider. The node counts the number of references to each index that it has, when the number exceeds a certain threshold; a replica of the content is placed instead of index at this moment. This approach is very simple but it is more efficient to reduce network traffic and increase search success ratio than reduce download time.

Paper [12] presents a solution for the problem of load balancing in peer-to-peer file sharing systems, based on the automated replication of files into “good” peers. Among the possible candidates to host a replica the best ones according to their bandwidth and connectivity is chosen. This approach considers appropriate factors to choose replica host.

Requester replication and path replication have been used by Gnutella and Freenet systems. In requester replication strategy, when a search is successful, the object is replicated and stored at the requester node only. In path replication strategy, when a search succeeds, the object is stored at all nodes along the path from requestor node to the provider node. In fact Gnutella implements passive replication, a file is only replicated at nodes requesting the file whereas Freenet allow proactive replications, where the object may be replicated at a node though the node has not requested the object [13].

3 Our Replication Method

In this work we focus on unstructured P2P networks like Gnutella. Data shared here assumed to be read only. In recent P2P file sharing systems the data is also read only. In this way the users share files such as music, films, books, etc, which won't change in future.

The proposed solution for replication is based on popularity of files. Peers measure the file popularity by counting the number of requests. When a peer receives a request for a file that it is able to provide it (or its index), it increases popularity of that file. In this work we try to decrease load of popular files provider nodes by replicating a copy of them in other nodes, and try to increase successful search rate of middle popular files by replicating their index in other nodes. So we consider two thresholds T1 and T2 for popularity (T1<T2).

The replication mechanism is divided into four important subjects: which file should be replicated, when to perform replication, where the new replicas should be placed and how replicas are destroyed (replica replacement policy).

3.1 What Should be Replicated?

Allocating more replicas of more popular contents brings improvement of search success as a whole because popular contents are more often searched. It must be also a benefit to reduce network traffic because the number of hops to reach a searched content becomes small [11].

The size of an index (Addresse of requester and provider) is much smaller than that file; therefore the cost of keeping an index is much smaller than the one of copying and storing the file.

3.2 When Should be Replicated?

When a peer receives a request for a file that is able to provide it (or its index), will increase popularity of that file, then it checks, if popularity of that file exceeds some threshold in a period, the peer will decide to replicate the file (or its index).

3.3 Where Should be Replicated?

Our method for replication is to simply replicate a file (or its index) along the path that a query has followed from requester to provider.The middle node located between requester and provider of a file is the best place to replicate its index, because the sum of hops from index will be minimized [11]. Fig. 1 shows the reason why the halfway node is the best to place a replica. The sum of the hop numbers will be minimized if a replica is placed on the node halfway the search path. This is more effective than probable allocation.

In an ideal world, every user shares or downloads the files, stays connected to the P2P network for a long time, in practice these systems have many problems like having a large number of low bandwidth or low connectivity peers. Low bandwidth peers, such as modem users, are not able to act as servers and are not able to let other peers download their files [12]. Low connectivity peers are peers that stay connected to the P2P network only for short periods at a time. These peers usually connect to the system to

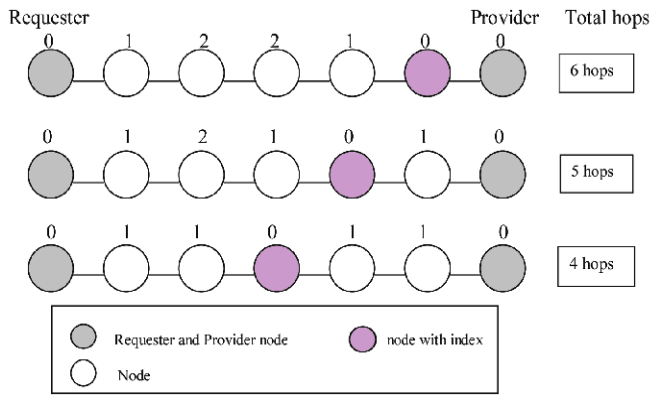

Fig. 1. Index replication on search path

download a few files and then disconnect. Measurements show that low bandwidth peers, stay connected to the P2P network for a shorter time [14]. Therefor among the candidate nodes (the nodes between provider and requester), the best for hosting replica is chosen based on bandwidth.

When a node receives a request that it is able to provide it, investigates the requests history for the requested file, if the number of requests exceeds T2 then a copy of file will be replicated in another node, but if the number of requests is more than T1 and less than T2 an index of requested file will be replicated in another node.

3.4 Replica Replacement Policy

Storage of peers is limited to keep replicated files or indexes so a replica replacement policy is essential to decisde which stored replicas should be replaced with the new replica in the shortage of storage spase at the selected node. We use Least Recently Used (LRU) to delete the replicas or indexes. In fact we use temporary locality used in Gnutella network.

The complete replication algorithm is given in Fig. 2.

```
if each peer P receives query q for item i from R and TTL >= 1 then{
  if P contains i then {
    R Accesses i from P;
    count of accesses for i in P increase 1;
    if count of accesses for i >= T2 then {
      s1= the peer is between R and P that it has the best bandwidth;
      if size of i <= s1 available storage size then {
            replicate a copy of i in s1;
          exit;
      }//end if
      else {
        delete some files from s1's replica storage using LRU;
        replicate a copy of i in s1;
        exit;
        }//end else
    }// end if
    else if T1<= count of accesses for i <T2 then{
        s2= the peer is halfway between R and P;
        if s2 has enough index storage then{
            replicate an index of i in s2;
          exit;
        }//end if
        else {
            delete an index from index storage of s2 using LRU;
            replicate an index of i in s2;
            exit;
        }//end else
    }// end else if
  }//end if
  else if an index of i is available in P then {
      count of accesses for i in P increase 1;
      if access count of index >= T2 then{
          replicate a copy of i from P, instead index;
      }//end if
  }//end elseif
  else{
    forward the q to all neighbors except the node which sent the query;
    TTL = TTL – 1;
  }//end else
}//end if
```

Fig. 2. Replication Algorithm

4 Simulation Setup

To simulate our proposed replication method we used C# language programming with SQL database. Our simulator is a Query-Cycle simulator. A simulation process proceeds in a query cycle. In each query cycle, a peer i in the network may be actively issuing a query. Upon issuing a query, a peer waits for incoming responses, selects a download source among those nodes that respond and starts downloading the file. The query cycle finishes when all peers who have issued queries download a satisfactory response. Statistics may be collected at each peer, such as popularity of requested file.

4.1 Network Model

We assume a pure Peer-to-Peer model like Gnutella and ignore physical connectivity.

In a decentralized unstructured network like Gnutella, each node usually connects to 3 other nodes. For simplicity like [15] we also use a mesh n×n structure to model network topology because the average of mesh connections (neighbors of each node) is between 3 and 4. We assume that nodes and their connections are fixed and will never change during simulation. We consider a mesh with 50×50 dimensions that it contains 2500 nodes.

4.2 Content and Query Distribution

For simplicity like [9], [12], we also consider all files the same size (4MB). In the first of simulation we have 6000 files, disseminated among nodes randomly. Like [11] we consider five levels for file popularity. It has been observed in P2P file sharing systems that queries follow a zipf distribution [9]. According to this distribution, files with more popularity are more searched. File popularity levels and their request rate shown in Table 1 which follows zipf distribution. According to these levels thresholds are as follows T1=3 and T2=5, Thresholds for algorithms in [11], [12] are also 3 and 5 respectively.

TABLE 1
POPULARITY LEVEL AND REQUEST RATE OF FILES

Popularity	Request rate (%)	Number of Files
(Low) 1	5	3500
2	10	1300
(Middle) 3	20	700
4	30	350
(High) 5	40	150

4.3 Bandwidth

We have a simple understanding of a peer's bandwidth in our simulations: Bandwidth at a peer is consumed only while uploading or downloading files. Bandwidth is assigned to peers based on measurements in [14]. These observations show that 8% of Gnutella users connect with dial up (64 kbps or less), 60% of users connect by broad-

band connections (cable, DSL, T1, and T3) and 30% also have high bandwidth connections (3Mbps and more). Fig. 3 shows the empirical CDF used in the simulations to model the bottleneck bandwidth of the peers. The chosen CDF reflects real life bottleneck bandwidths observed in Gnutella [14].

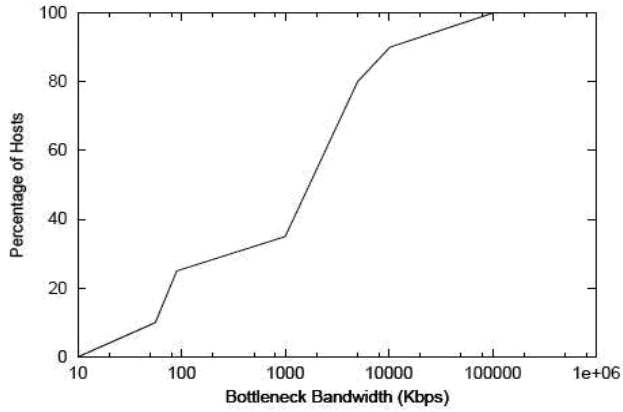

Fig. 3. CDF of bottleneck bandwidths for Gnutella peers

We assume that each peer has a limited bandwidth capacity to download and upload files. Download and upload capacity is assumed to be 1. That means there is only one download or upload operation that can be executed at a time.

Table 2 presents the values of the other parameters used in the simulations.

TABLE 2
SIMULATION PARAMETERS

Simulation Parameter	Value
TTL	5
Maximum count of replicas that each node can holds (Replica Storage)	100
Maximum count of indexs that each node can holds (Index Storage)	1000
Number of queries in every experiment	100,000

5 SIMULATION RESULTS

To evaluate our proposed replication strategy, we choose these performance metrics: average download time, average of hops for search and success rate of queries. We compare our strategy with requester strategy and proposed strategies in [11], [12] because we believe that our strategy enhances them. We also present the costs for all of these strategies.

Fig. 4 shows average download time for mentioned strategies. Average download time is calculated at every query cycle. It's clear that our strategy has a better performance over other strategies.

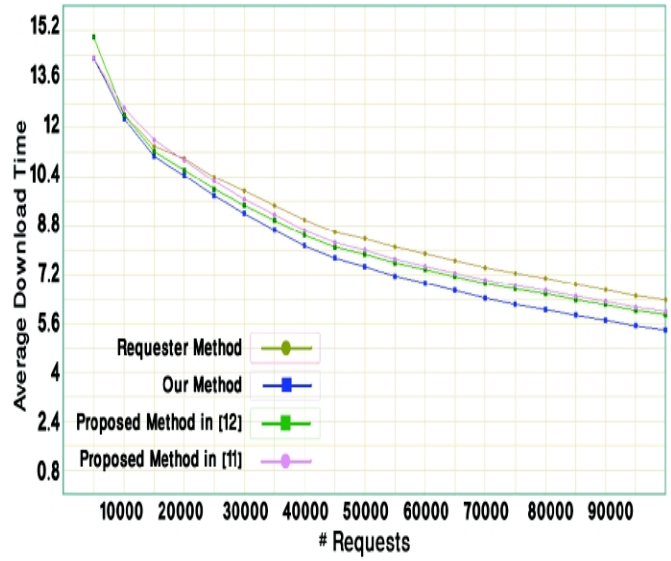

Fig. 4. Average Download Time

Fig. 5 shows successful searches. If a requested data exist and it can be found then that search is successful otherwise unsuccessful.

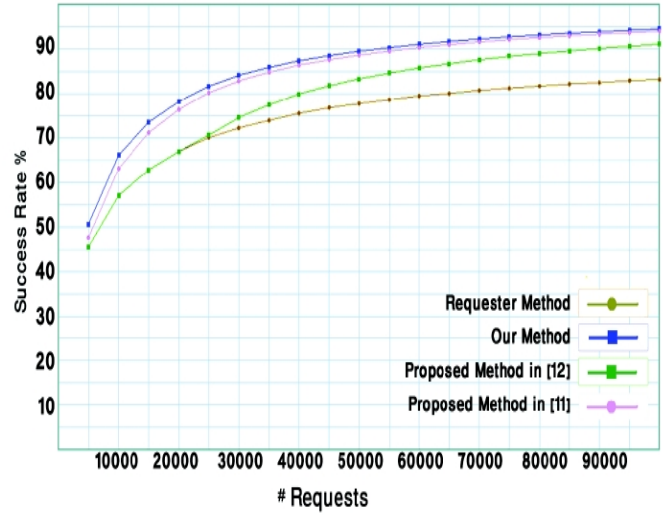

Fig. 5. Success Rate of Search

Fig. 6 shows average of search hops. When a node requests a file, if the node itself has the file the number of hops is equal with zero, if file is found in another node then the number is equal with sum of visited nodes between requester and provider plus 1and if the search is unsuccessful the number of hops will equal TTL. As the Fig. 5 and Fig. 6 show, our strategy works better than others and strategy [11] works better than [12] because strategy [11] focuses on improvement of search efficiency by indexing replication but strategy [12] tries to improve average download times by replicating the popular files in high bandwidth nodes.

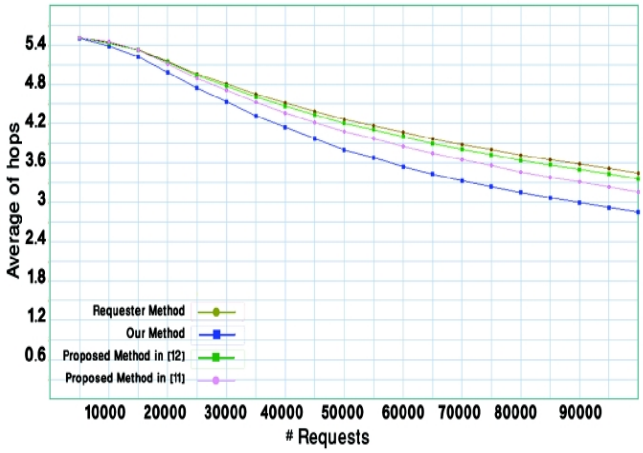

Fig. 6. Average of Hops

Fig. 7 shows average number of replicated files. We present this chart to evaluate the cost of replication for our strategy. It's clear the more the number of replicated files the more the cost of storage. As the Fig. 7 shows, our strategy has worst cost but difference of maximum point of our strategy's curve and green curve is a small value, Threfore cost of our strategy is not so much more than [11], [12] startegies.

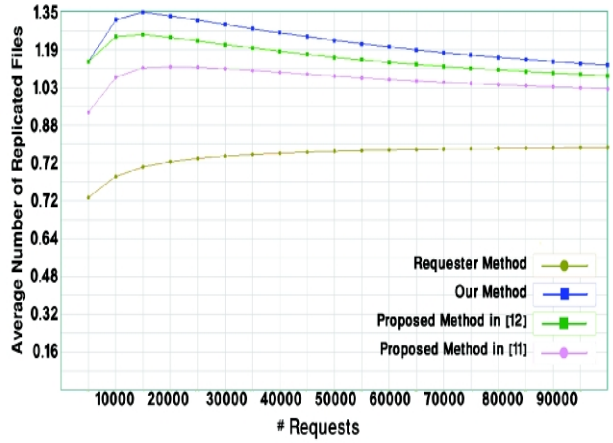

Fig. 7. Average Number of Replicated Files

6 Conclusion Conclusion and Future Works

In this paper, we proposed a pro-active novel strategy for data replication in pure decentralized unstructured P2P systems. Our approach can be used in unstructured P2P networks such as Gnutella.

This method is decentralized and combines data replication with index replication and uses two thresholds for popularity.

Through the simulation results, this strategy has a better performance in terms of success rate, average download time and average number of search hop compared with other strategies. Although, the proposed strategy might use more storage for the replication of index, the overhead is considered to be small compared to the benefits achieved from its performance.

We tested our strategy with flooding search. There are some other methods for search in P2P systems such as random walker and expanding ring so Future works are as follows. First, respect of popularity of file and its number of replicas must be investigated. Second, other P2P search methods must be examined as well as flooding. Third, we are going to expand proposed strategy to use in structured P2P systems.

References

[1] Lv, Q., Cao, P., Cohen, E., Li, K., and Shenker, S., "Search and replication in unstructured peer-to-peer networks" In Proceedings of the 16th annual ACM International Conference on Supercomputing, 2002.

[2] Napster, http://www.napster.com. Retrieved August 17, 2008.

[3] Gnutella protocol specification v0.4. Retrieved August 17, 2008, fromhttp://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

[4] Y. Saito and M. Shapiro. "Optimistic replication". ACM Computing Surveys, 37(1):42 – 81, Mar. 2005.

[5] Mauro Sozio, Thomas Neumann, and Gerhard Weikum. "Near-optimal dynamic replication in unstructured peer-to-peer networks". In Proceedings of the 27th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2008), pages 281-290, 2008.

[6] Venugopalan Ramasubramanian and Emin G□un Sirer. Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays. In Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI 2004), pages 99-112, 2004.

[7] Ben Leong, Barbara Liskov, and Erik D. Demaine. Epichord: Parallelizing the Chord lookup algorithm with reactive routing state management. Computer Communications, 29(9):1243-1259, 2006.

[8] Ranganathan, K. and Foster, I. T. (2001). "Identifying dynamic replication strategies for a high-performance data grid". In Proceedings of the International Grid Computing Workshop, pages 75–86.

[9] Edith Cohen and Scott Shenker. "Replication strategies in unstructured peer-to-peer networks". In Proceedings of the ACM SIGCOMM 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, pages 177-190, 2002.

[10] Dimitrios Tsoumakos, Nick Roussopoulos, " APRE: A Replication Method for Unstructured P2P Networks", 2007.

[11] Kawasaki, Y., Matsumoto, N. and Yoshida, N., "Popularity-Based Content Replication in Peer-to-Peer Networks", V.N. Alexandrov et al. (Eds.): ICCS 2006, Part IV, LNCS 3994, pp. 436–443, 2006.

[12] L. Abad, C., "Load Balancing through Automated Replication in Unstructured P2P File Sharing Systems", [JBCS]- Journal of the Brazilian Computer Society, ISSN 0104-6500, 2007

[13] Martin, V., Pacitti, E., Valduriez, P., "Survey of Data Replication in P2P systems", inria-00122282, version 2, 2007

[14] Saroiu, S., Gummadi, P. K., and Gribble, S. D , "A Measurement Study of Peer-to-Peer File Sharing Systems", In Proceedings of the Multimedia Computing and Networking, 2002

[15] M. Karakaya, I. Korpeoglu, O. Ulusoy, "GnuSim: a Gnutella network simulator", in: Technical Report BU-CE-0505, Department of Computer Engineering, Bilkent University, 2005