Efficient and Effective File Replication in Structured P2P File Sharing Systems

Haiying Shen
Department of Electrical and Computer Engineering
Clemson University, Clemson SC 29634
shenh@clemson.edu

Abstract

In peer-to-peer file sharing systems, file replication helps to avoid overloading file owners and improve file query efficiency. Aiming to achieve high replica utilization and efficient file query with low overhead, this paper presents a file replication mechanism based on swarm intelligence, namely SWARM. Recognizing the power of collective behaviors, SWARM identifies node swarms with common node interests and close proximity. It determines the location of a file replica based on the accumulated query rates of nodes in a swarm. Replicas are shared by the nodes in a swarm, leading to less replicas and high query efficiency. Simulation results demonstrate the efficiency and effectiveness of the SWARM mechanism in comparison with other file replication methods.

1 Introduction

Over the past years, the immense popularity of Internet has produced a significant stimulus to peer-to-peer (P2P) networks. One of the most popular applications of P2P networks is file sharing such as BitTorrent [1], KaZaA [8] and Morpheus [9]. In such systems, if a node receives a large volume of requests for a file at one time, it would become a hot spot, leading to delayed response.

File replication is an effective strategy to deal with the problem of overload due to hot files. It replicates a hot file to other nodes to distribute the load, and improve file query efficiency by reducing query latency. Current file replication methods mainly can be classified into three classes denoted by *ServerEnd* [13, 4, 16], *Path* [12, 19, 3] and *ClientEnd* [6, 5]. Recently, we proposed Efficient and Adaptive Decentralized file replication algorithm (*EAD*) [14]. *EAD* selects the query traffic hubs that receive many queries of a file and frequent requesters as replica nodes. However, these methods cannot guarantee that a file query encounters a replica node when the file has a replica. To further enhance the efficiency and effectiveness of file replication, this paper presents a file replication mechanism based on swarm intelligence, namely SWARM. Swarm intelligence is the property of a system whereby the collective behaviors of agents cause coherent functional global patterns [2]. Recognizing the power of collective behaviors, SWARM identifies node swarms with common interests and close proximity for file replication and replica sharing.

2 Related Work

As mentioned above, previous file replication methods generally can be classified into three categories: *ServerEnd*, *ClientEnd* and *Path*. In the *ServerEnd* category, PAST [13] replicates each file on a set number of nodes whose IDs match most closely to the file owner's ID. It has load balancing algorithm for non-uniform storage node capacities and file sizes. Similarly, CFS [4] replicates blocks of a file on nodes immediately after the block's successor on the Chord ring [17]. Stading *et al.* [16] proposed to replicate a file in proximity-close nodes near the file owner. In the *ClientEnd* category, LAR [6] and Gnutella [5] replicate a file in overloaded nodes at the file requesters. Backslash [16] proposed to push cache to one hop closer to requester nodes as soon as nodes are overloaded. In the *Path* category, CFS [4], PAST [13], LAR [6], CUP [12] and DUP [19] perform caching along the query path. Cox *et al.* [3] studied providing DNS service over a P2P network. They cache index entries, which are DNS mappings, along search query paths. *EAD* [14] enhances the utilization of file replicas by selecting query traffic hubs and frequent requesters as replica nodes, and dynamically adapting to non-uniform and time-varying file popularity and node interest.

3 Efficient and Effective File Replication

Figure 1 shows the basic idea of the SWARM mechanism based on swarm intelligence. The nodes $a-h$, p and s have the same interest "book". Node s is the owner of a "book" file, and other nodes frequently request the file. SWARM

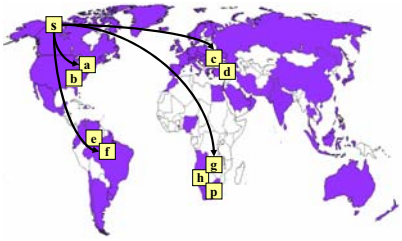


Figure 1. File replications in SWARM.

forms common-interest and proximity-close nodes into a node swarm, and makes one replica for each swarm. The swarms with the same interest further constitute a node colony. A replica is shared by all nodes in a swarm and by all swarms in a colony. Thus, SWARM reduces replicas and replication overhead, and enhances replica utilization.

SWARM builds the swarm structure using a landmarking method [15, 11, 18] that represents node closeness on the network by indices. Landmark clustering is based on the intuition that nodes close to each other are likely to have similar distances to a few selected landmark nodes. Sophisticated strategies [10] can be used for landmark node selection. We assume m landmark nodes are scattered in the Internet. Each node measures its physical distances to the m landmarks, and uses the vector of distances $< d_1, d_2, ..., d_m >$ as its landmark vector. Two physically close nodes have similar landmark vectors. Hilbert curve [18] is then used to map m-dimensional landmark vectors to real numbers, such that the closeness relationship among the nodes is preserved. The number is called the *Hilbert number* of the node, denoted by $\mathcal{H}$, that indicates the physical closeness of nodes on the Internet. Two physically close nodes have close $\mathcal{H}$s.

Without loss of generality, we assume that a node's interests are described by a set of attributes described by globally known strings such as "image", "music" and "book". Each interest corresponds to a category of files. Using consistent hash function [7], it is computationally infeasible to find two different messages that produce the same message digest. Therefore, the hash function is effective to group interest attributes. Same interest attributes will have the same consistent hash value, while different interest attributes will have different hash values.

SWARM uses the Hilbert number and consistent hash function to build node swarms based on node interest and physical proximity. To facilitate such structure construction, the information of physically close nodes with a common interest should be marshaled in one node in the DHT network, which facilitates these nodes to locate each other to constitute a swarm. Although logically close nodes may not have common interest or be physically close to each other, SWARM enables common-interest nodes to report their information to the same node, which further clusters the information of proximity-close nodes into a group.

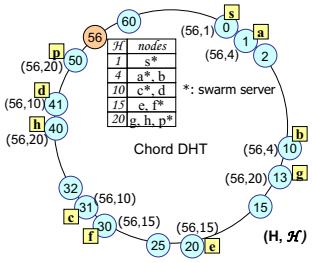


Figure 2. Information marshaling for swarm construction. *By* `Insert(H,Info)`*, the information of nodes with a common interest is marshalled in node 56, which further clusters the information of the same $\mathcal{H}$ into a group.*

In a DHT overlay, an object with a DHT key is allocated to a node by the interface of `Insert(key,object)`, and the object can be located by `Lookup(key)`. If two objects have the same key, they are stored in the same node. We use H to denote the consistent hash value of a node's interest, and $Info$ to denote the information of a node:

$$Info =< \mathcal{H}, IP, ID >,$$

where IP and ID are the IP address and ID of the node. Because H distinguishes node interests, if nodes report their information to the DHT with their H as the key by `Insert(H,Info)`, the information of common-interest nodes with the same H will reach the same node, which is called *repository node*. The repository node further clusters the information with the same $\mathcal{H}$ into a group. As a result, a group of information in a repository node is the information of physically close nodes with a common interest. Figure 2 shows an example of information marshaling in Chord with the nodes in Figure 1. These nodes have interest "book" and $H(book) = 56$. 2-tuple notation such as (56,4) in the figure represents $(H, \mathcal{H})$ of a node. The nodes send their information with their $H = 56$ as key using `Insert(56,Info)`. All of the nodes' information will arrive at node 56 which further groups the information based on $\mathcal{H}$ as shown in the table. Consequently, the information of physically close nodes with a common interest is clustered in one group in the repository node.

If a node has a number of interests, it reports its information based on each interest. When a node becomes very interested in a file category, it reports its information to a repository node based on this interest. When the node is no longer interested in a file category, it asks the repository node of the interest to remove its information. By this way, SWARM mechanism tunes to time-varying node interest dynamically, and a node colony always consists of frequent requesters.

Therefore, a node can find other physically close nodes with the same interest in its repository node by `Lookup(H)`. After these nodes discover each other, they

constitute a swarm. All swarms of a interest constitute a colony. Specifically, in each swarm, the highest-capacity node is elected as the *server* of other nodes (i.e. *clients*) in the swarm. Swarm servers are responsible for file query among swarms in a colony. A server maintains an index of all files and file replicas in its clients.

Rather than replicating a file in individual requesters, SWARM considers the request frequency of a node swarm, and makes replicas for the swarm. We define a requester's *query rate* of file f, q_f, as the number of queries for file f that the requester initiates during a unit time, say one second. The rate calculation method based on exponential moving average technique [14] can be used to reasonably determine the query rate. We define *swarm query rate* of file f, s_f, as the number of f queries initiated by the nodes in a swarm during one second. Physically close nodes have the same $\mathcal{H}$, so that a requester includes its $\mathcal{H}$ in its file request in order to facilitate computing s_f.

In addition to file owners, SWARM enables replica nodes to replicate their replica files. A node periodically calculates a file requester's q_f for a file and s_f based on $\mathcal{H}$. That is:

$$(s_f = \sum q_{f_i} | \mathcal{H}_i = v),$$

where v is a specific value of $\mathcal{H}$. When overloaded, the node replicates the file in the most frequent requester in the swarm with the highest s_f. This arrangement is to increase the replica utilization by making it shared by more frequent requesters. Specifically, the node chooses the swarm with the highest s_f, then orders the swarm nodes in descending order of q_f, and selects a non-replica node in a top-down fashion. The replica node will report to its server of its new replica.

When node i requests for a file, if the file is not in the requester's interests, the node uses DHT `Lookup(key)` function to query the file. Otherwise, node i first queries the file in its swarm among physically close nodes interested in the file, and then in its colony among nodes interested in the file. Specifically, node i first sends a request to its swarm server of the interest. The server searches the index for the requested file in its swarm. If the searching fails, the server queries for the file replica in a nearby swarm.

4 Performance Evaluation

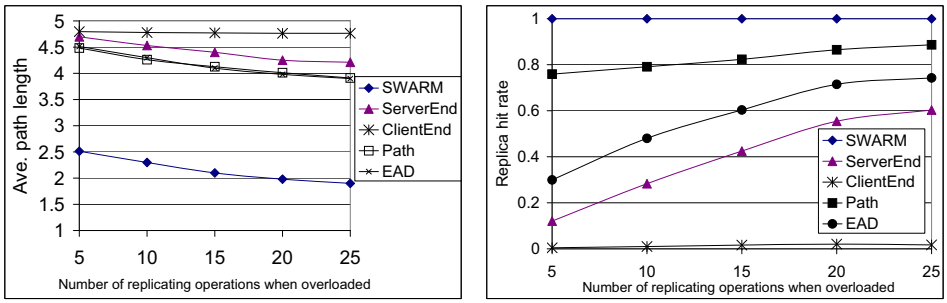
This section presents the performance evaluation of SWARM in comparison with *ClientEnd*, *ServerEnd*, *Path* and *EAD* in file replication. We use *replica hit rate* to denote the percentage of the number of queries among total queries that are resolved by replica nodes. In the experiment, when overloaded, a node conducts a file replicating operation. In one file replicating operation, SWARM, *ServerEnd*, *ClientEnd* and *EAD* replicate a file to one node while *Path* replicates a file to a number of nodes along a query path. The number of nodes was set to 2048. We assumed there were 200 interest attributes, and each attribute had 500 files. We assumed a bounded Pareto distribution for the capacity of nodes. Table 1 lists the parameters of the simulation and their default values, unless otherwise specified.

Table 1. Simulated environment and parameters.

Parameter	Default value
Object arrival location	Uniform over ID space
Node capacity c	Bounded Pareto: shape 2 lower bound: 500 upper bound: 50000
Number of queried files	50
Number of queries per file	1000
α	0.5
$T_{s_f}/T_s/d$	8/10/2

Figure 3(a) plots the average path length of different approaches. We can see that *Path* and *EAD* generate shorter path length than *ServerEnd* and *ClientEnd*, and SWARM leads to the shortest path length. Unlike others that replicate a file only in one node in each file replicating operation, *Path* replicates file in nodes along a query path. Therefore, it increases replica hit rate and produces shorter path length. However, it is unable to guarantee that every query can encounter a replica. By replicating a file in traffic hubs that forward many queries of the file, *EAD* leads to high hit rate and short path length. The result that SWARM achieves much higher lookup efficiency confirms its effectiveness in replicating files for a group of common-interest and proximity-close nodes based on their accumulated query rate. A node can get a file directly from a node in its own swarm which enhances the utilization of replicas and also reduces the lookup path length. *ServerEnd* replicates a file close to the file's owner, such that the file's request will encounter a replica node before arriving at the owner, shortening lookup path length. However, since the replica nodes locate close to the file owner, the requests need to travel more hops than in other methods. Therefore, it is not able to significantly reduce lookup path length. Surprisingly, *ClientEnd* generates much longer lookup path length than others. This is because files are replicated in requesters, and requests from other nodes may not pass through the replica nodes with high probability. Consequently, *ClientEnd* is not able to make full use of file replicas to reduce lookup path length. In contrast, SWARM enables a replica be shared within a group of frequent requesters, which dramatically increases the utilization of replicas and reduces path length.

Figure 3(b) demonstrates the replica hit rate of different approaches. We can observe that *ClientEnd* generates the least hit rate, *Path* leads to higher hit rate than *EAD*, *ClientEnd* and *ServerEnd*, and SWARM leads to the highest hit rate. For the same reason observed in Figure 3(a), *Clien-*



(a) Ave. path length

(b) Replica hit rate

Figure 3. Efficiency and effectiveness of file replication approaches.

tEnd has very low replica hit rate. *ServerEnd* replicates a file near its owner, such that a query for the file can encounter a replica node before it arrives at the owner with high probability. *EAD* replicates a file in traffic hubs that forward many queries of the file. Therefore, EAD produces high probability that many queries meet replica nodes, leading to high hit rate. *Path* replicates files at nodes along the routing path, more replica nodes render higher possibility for a file request of meeting a replica node. However, its efficiency is outweighed by its high overhead of replicating much more replicas. In addition, it cannot ensure that each request can be resolved by a replica node. SWARM replicates a file for a group of common-interest nodes and enables a node to actively retrieves the locations of replica nodes, which significantly improves the probability that the file query is resolved by a replica node, leading to much higher hit rate.

5 Conclusions

Current file replication methods for P2P file sharing systems are not sufficiently effective in improving file query and replica utilization. This paper proposes a swarm intelligence based file replication called SWARM. SWARM builds common-interest and proximity-close nodes into a swarm, and relies on super nodes to connect swarms with the same interest into a colony. It replicates a file in a swarm with the highest accumulated file query rates of the swarm nodes, and makes the replica being shared among the nodes in a swarm and colony. Simulation results demonstrate the superiority of SWARM in comparison with other approaches. It reduces the overhead of file replication, and produces significant improvements in lookup efficiency and replica hit rate.

Acknowledgment

The authors are grateful to the anonymous reviewers for their valuable comments and suggestions. This research was supported in part by U.S. NSF grants CNS-0834592 and CNS-0832109.

References

[1] Bittorrent. http://en.wikipedia.org/wiki/Bittorrent.
[2] Swarm Intelligence. http://www.sce.carleton.ca/netmanage/tony/swarm.html.
[3] R. Cox, A. Muthitacharoen, and R. T. Morris. Serving DNS using a Peer-to-Peer Lookup Service. In *Proc. of IPTPS*, 2002.
[4] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proc. of SOSP*, 2001.
[5] Gnutella home page. http://www.gnutella.com.
[6] V. Gopalakrishnan, B. Silaghi, and et al. Adaptive Replication in Peer-to-Peer Systems. In *Proc. of ICDCS*, 2004.
[7] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *Proc. of STOC*, 1997.
[8] Kazaa. Kazaa home page: www.kazaa.com.
[9] Morpheus home page. http://www.musiccity.com.
[10] T. S. E. Ng and H. Zhang. Towards global network positioning. In *Proc. of the 1st ACM SIGCOMM Workshop on Internet Measurement*, 2001.
[11] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *Proc. of INFOCOM*, 2002.
[12] M. Roussopoulos and M. Baker. CUP: Controlled Update Propagation in Peer to Peer Networks. In *Proc. of USENIX*, 2003.
[13] A. Rowstron and P. Druschel. Storage Management and Caching in PAST, a Large-scale, Persistent Peer-to-Peer Storage Utility. In *Proc. of SOSP*, 2001.
[14] H. Shen. An efficient and adaptive decentralized file replication algorithm in p2p file sharing systems. *TPDS*, 2009.
[15] H. Shen and C.-Z. Xu. Hash-based Proximity Clustering for Efficient Load Balancing in Heterogeneous DHT Networks. *JPDC*, 2008.
[16] T. Stading, P. Maniatis, and M. Baker. Peer-to-peer Caching Schemes to Address Flash Crowds. In *Proc. of IPTPS*, 2002.
[17] I. Stoica, R. Morris, D. Liben-Nowell, and et al. Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *TON*, 1(1):17–32, 2003.
[18] Z. Xu and et al. Turning Heterogeneity into an Advantage in Overlay Routing. In *Proc. of INFOCOM*, 2003.
[19] L. Yin and G. Cao. DUP: Dynamic-tree Based Update Propagation in Peer-to-Peer Networks. In *Proc. of ICDE*, 2005.