Distributed Caching and Adaptive Search in Multilayer P2P Networks

Chen Wang, Li Xiao, Yunhao Liu, Pei Zheng
Department of Computer Science and Engineering
Michigan State University, East Lansing, MI 48824
{wangchen, lxiao, liuyunha, zhengpei}@cse.msu.edu

Abstract

To improve the scalability of Gnutella-like unstructured Peer-to-Peer (P2P) networks, a uniform index caching (UIC) mechanism was suggested in some earlier work. In UIC, query results are cached in all peers along the inverse query path such that the same query of other peers can be replied from their nearby-cached results. However, our experiments show that the UIC method causes a large amount of duplicated and unnecessary caching of items among neighboring peers. Aiming at improving the search efficiency, we propose a distributed caching *mechanism which distributes the cache results among neighboring peers. Furthermore, based on the distributed caching mechanism, an* adaptive search *approach is built which selectively forwards the query to the peers with a high probability of providing the desired cache results. All the enhancements above are defined in a protocol called Distributed Caching and Adaptive Search (DiCAS). In the DiCAS enhanced Gnutella network, all the peers are logically divided into multiple layers, with the character that all the peers in the same layer have the same group ID. The query flooding is restricted in one layer with the matched group ID. Our simulation study shows that, with the help of the index caching and search space division, the DiCAS protocol can significantly reduce the network search traffic in unstructured P2P systems without degrading query success rate.*

1. Introduction

Compared with a structured P2P network[1-4], an unstructured P2P network is less efficient due to its blind flooding search mechanism. However, The unstructured P2P system, such as Gnutella [5], still gains high popularity in today's Internet community because of its simplicity. In Gnutella-like P2P system, a query is broadcast and rebroadcast until a certain criterion is satisfied. If a peer receiving the query can provide the requested object, a response message will be sent back to the source peer along the inverse of the query path.

The Breadth First Search behavior in a Gnutella system causes exponentially increased network traffic. Measurements in [6] show that even given that 95% of any two nodes are less than 7 hops away, the message time-to-live (TTL=7) is preponderantly used, the flooding-based routing algorithm generates 330 TB/month in a Gnutella network with only 50,000 nodes, in which 91% of the traffic was query messages and 8% was PING messages. Studies in [7] and [8] show that P2P traffic contributes the largest portion of the Internet traffic based on their measurements on some popular P2P systems, such as FastTrack (including KaZaA and Grokster) [9], Gnutella and DirectConnect. The inefficient blind flooding search technique causes the unstructured P2P systems being far from scalable [10].

Many efforts have been made to avoid the large volume of unnecessary traffic incurred by the flooding-based search in unstructured P2P systems. One approach is to change the flooding search behavior. For example, k-walker [11] tries to avoid the exponential increase of flooding traffic by selecting only several search paths among the peers. In order to compensate for the possible missing of peers in each query, replication strategies are also suggested in [11]. Another method is topology optimization [12], which intends to structure Gnutella-like P2P networks. In the supernode P2P network, each leaf peer is connected to a supernode that maintains all the indices such that query search within supernodes is sufficient to find the shared objects in the whole system. The overhead is the index update between the leaves and supernodes. The cluster based topology tries to limit search in a small space by dividing the whole network into multiple clusters. SIL[13] points out that a parallel search cluster based P2P network is superior to a supernode network in many aspects, such as robustness and load balance. The third mechanism is to employ some forms of cache or replication, in which file contents or query response results can be cached in non-query peers in hoping that future queries can be satisfied within a short query distance. Observations in [14] show that queries in a Gnutella network obey Power-law distribution, which significantly benefits the cache mechanism since the popular keywords are repeatedly queried. Therefore, a uniform index caching (UIC) mechanism is suggested in [14], which caches query results in all peers along the inverse query path. Compared with the supernode network, the overhead of index maintenance in the

---

This work was partially supported by the US National Science Foundation (NSF) under grant ACI-0325760, by Michigan State University IRGP Grant 41114.

cache method is small since they are carried by query responses in the way of a free ride. There is a trade-off between the extra cache storage in each peer and the search efficiency.

In the first part of this study, we implemented an Index Cache-enabled Gnutella Client (CGC) to cache query results in a real P2P network, measured query patterns and observed the cached results for the purpose of examining the caching efficiency. Even a single index cache-enabled peer connected to the Gnutella network can contribute a 20% cache hit ratio, which means 20% of queries can be replied using the cache results instead of being further forwarded. However, our experiments also show that a uniform index caching mechanism can easily cause a large amount of duplicated and unnecessary caching results among neighboring peers. Aiming to improve the search efficiency, we propose a *distributed caching* mechanism which distributes the cache results among neighboring peers. Furthermore, in a P2P system enabled with distributed index caching, since a query result will be only cached in selected peers, there is no need to flood the query to all peers. Instead, we propose an *adaptive search* mechanism which selectively forwards the query to only peers with a high probability of providing the desired cache results. Since only a portion of the peers among the neighbors are selected during each forwarding process, the volume of search traffic is reduced significantly. All the enhancements above are defined in the Distributed Caching and Adaptive Search (DiCAS) protocol. In DiCAS, each node randomly takes an initial value in a certain range [0..M-1] as a group ID when it participates into the P2P system. We define that a query matches a peer if and only if the equation below is satisfied:

$$\textit{Peer Group ID} = \textit{hash(query) Mod } M$$

Under the DiCAS protocol, a query response will only be cached in a matched peer. The query forwarding will also be restricted to matched peers. In the DiCAS enhanced Gnutella P2P network, the group of all peers are divided into multiple layers, with the character that the peers in the same layer have the same group ID. The query flooding is restricted within one layer with the matched group ID. Figure 1 shows an example when M equals 3.

Our simulations have shown that the DiCAS protocol can significantly reduce the network traffic incurred by search in unstructured P2P systems without degrading the query success rate.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 presents our implementation of Index Cache-enabled Gnutella Clients, and experimental results on the index cache-enabled clients connecting in a real P2P network. Section 4 describes the Distributed Caching and Adaptive Search scheme. Section 5 describes our simulation methodology. Performance evaluation of the DiCAS is presented in Section 6, and our study is concluded in Section 7.

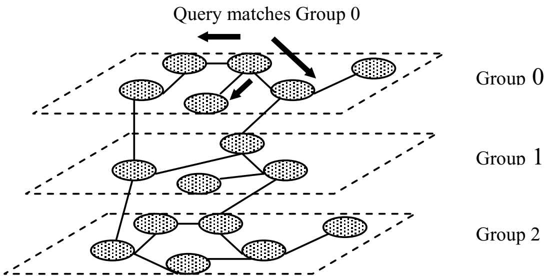

Figure 1 Flooding in multilayer P2P network

2. Related Work

Several mechanisms have been proposed to improve search efficiency in decentralized unstructured P2P systems.

The authors in [14] analyze the characteristics of Gnutella queries and their popularity distribution, and propose that each peer cache query strings and results that flow through it. Similar to [14], the authors in [15] observed that submitted queries exhibit significant amounts of locality based on one hour of Gnutella traffic, and proposed a caching mechanism that caches query responses according to the timestamp the query is responded to. The effectiveness of caching query results has been shown by simulations in both [14] and [15]. All the work above suggests a uniform index caching (UIC) mechanism. The UIC causes a large amount of duplicated and unnecessary cache results among neighboring peers, which is shown by our experiment.

Based on an observation of query locality in peers behind a gateway of an organization, transparent query caching [16] is proposed to cache query responses at the gateway. Contrast to the work in [16], our approach can fully take advantage of the internal nodes' resources and avoid the bottleneck in the centralized cache.

Caching file content has also been studied. An ideal cache (infinite capacity and no expiration) simulator is built [8] to investigate the performance of content caching for KaZaA P2P network. It has been shown that caching would have a great effect on a large-scale P2P system on reducing wide-area bandwidth demands. Compared with the index caching, the content caching is less storage efficient.

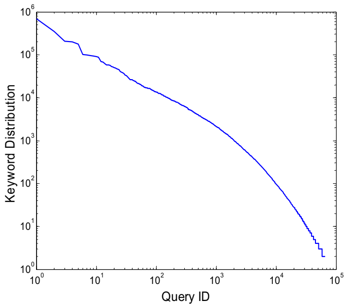

Figure 2 Keyword distribution of the CGC query trace. Query IDs are ordered by the frequency.

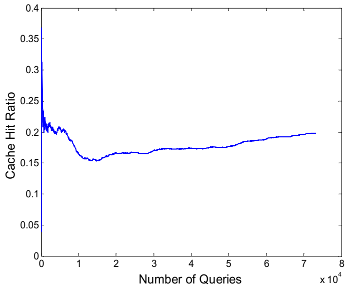

Figure 3 Cache hit ratio on a single CGC peer with trace-driven query generator. Cache size is 1024 KB.

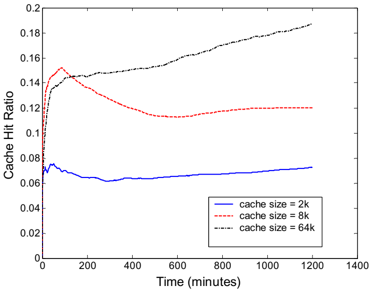

Figure 4 Cache hit ratio with different cache sizes in single CGC experiment.

The k-walker [11] proposes a random walk search mechanism and evaluates three different strategies to replicate data (file content or query responses) on multiple peers. Uniform strategy creates a fixed number of copies when the item first enters the system. Proportional strategy creates a fixed number of copies every time the item is queried. In square-root replication strategy, the ratio of allocations is the square root of the ratio of query rates. Our work is different from K-walker in that the query is more tightly connected to the cache. In DiCAS, the query is forwarded intentionally to the peers with a high probability to provide the desired cache results.

The superiority of the cluster based P2P network has been mathematically proved in [13], while the mechanism of how to break the P2P network into multiple clusters has not been mentioned yet.

3. Experiments of Index Caching in Gnutella Network

3.1. Overview of Experimental Setup

To investigate the performance impact of index caching in a real, large scale peer-to-peer network, we modified LimeWire Gnutella servant [17] with support of Gnutella protocol v0.6 [18], and developed an index Cache-enabled Gnutella Client (CGC). Compared with a normal Gnutella client, Our CGC peer is able to create and maintain a local index cache by overhearing traversing query response results in existing Gnutella network. As a result, other peers neighboring to the CGC peer will have the opportunity to utilize the index cache for future search. We conduct a number of experiments with the CGC peer in Gnutella network to explore the performance improvement by the index caching.

We have also built a traffic monitoring tool that works in conjunction with the CGC peer to trace incoming and outgoing queries and responses, as well as cache hits and misses on the index cache. Two aspects of the dumped P2P network traffic through the CGC peer could be used. First, by analyzing the query patterns and locality in both space and time we could gain more insight into some fundamental issues of index caching, such as how to determine the cache size and the cache expiration time. Second, the trace data can be used as traffic source to flexibly test our CGC peer implementation in a variety of scenarios with comparable results. In this sense, we have actually built a cache-aware P2P network testbed with the CGC experimental setup and the traffic monitoring and trace-driven tool. In the following section, we present four test scenarios that employ either single or multiple CGC peers to examine the benefit and overhead of index caching in a Gnutella network.

The CGC peer is a PC with a 2.4GHz Pentium IV processor, 1 Gigabytes memory, and Ethernet connection to the campus network. The CGC software is running on Linux. We use LRU as the index cache replacement policy. Other cache replacement policies can also be incorporated into CGC. Clearly, different cache replacement policies will have different effects on the hit ratio of the index cache. To examine the impact of cache size on overall performance, we vary the cache size from 2 Kbytes to 64 Kbytes.

3.2. Trace-driven Single CGC Peer Experiment

Our Gnutella network query trace was collected on one CGC peer on March 11, 2003. Some non-meaningful words such as articles and propositions have been removed from the trace to improve the accuracy of our analysis. The total number of queries is 13,705,339, while 129,293 unique keywords exist in the trace. As shown in Figure 2, the frequency of query keyword in the trace roughly follows a Zipf distribution, which substantially suggests that the index cache in a Gnutella network could make use of the keyword and query response result locality to improve searching performance. Figure 3 shows the index cache hit ratio on the single CGC peer in a trace-driven query experiment. The cache size is 1024Kbytes in this case. It shows that about 21% of total traversing queries will be replied by the single CGC index cache.

3.3. Single CGC Experiment

The single CGC is connected to Gnutella network, which works as a regular Gnutella client except that it maintains an index cache. Intuitively, the benefit of index caching will be more evident if the CGC peer could populate a large number of cached items. Hence, the CGC has been configured to be an ultrapeer that has a higher probability to establish connections with others than regular peers, according to Gnutella protocol. We have observed that the number of neighbors of the CGC peer ranges from less than 10 to 120.

Figure 4 illustrates the index cache hit ratio as a function of time with different cache sizes in one day, which clearly shows that if the cache size increases, the hit ratio will increase as well. We identified that the major factors that limit the hit ratio of an index cache are transit search locality of neighboring peers to the CGC peer, and the number of neighboring connections the CGC peer could reach over the time we conduct the experiment. We expect that with a longer warm-up time and allowing more connections to the CGC peer, the cache hit ratio will be further improved.

We also expect if there are more CGC peers participating in the P2P network, the overall searching performance can be improved due to cooperation between the neighboring peers. However, our following experiments show that a large amount of duplicated items are cached among neighbors.

3.4. Twin CGC Peer Experiment

For the purpose of investigating the performance of distributed index caching in a Gnutella network, we connected our two CGC ultrapeers into the network. The two CGC ultrapeers should be logical Gnutella neighbors in the overlay such that we could measure their overall contributions to traversing queries with two separated index caches. Due to the inherent overlay nature of P2P systems and Gnutella's topology optimization scheme, the two CGC ultrapeers, even if they are close to each other in the physical network, cannot maintain a persistent neighboring relation after some up time. To enforce a fixed neighboring relation between the two CGC ultrapeers (the Twin CGCs), a "dummy" regular Gnutella Client is added to the test environment. The "dummy" GC can only have two neighbors (the twin CGCs) in the overlay. It is "dummy" because when forwarding queries between the twin CGCs, it will not decrease TTL of the query such that the two peers connected by the dummy Gnutella Client become virtual neighbors.

In the Twin CGCs experiment, all the cache hits in each CGC were recorded in log files. The comparison between the records of both peers shows significant cache hit overlap between the neighboring peers. Among the 8500 cache hits recorded in each peer within two hours, there are 2741 duplicated cache hits. The overlapped cache hits between two neighboring peers exceed 32% of all the cache hits in one peer. We expect more overlap among neighboring peers when multiple peers are fully connected with each other. Those duplicated cache hits are unnecessary since only one of the duplicated cache hit is sufficient to satisfy the correspondent query. The observation above suggests that it is possible to improve the search efficiency by distributing index cache among neighboring peers. Based on the distributed index caching, the search efficiency can be further improved by our adaptive search method which forwards the query to only peers with the matched Group ID.

4. Distributed Caching and Adaptive Search

4.1. Gnutella Protocol

We first briefly introduce a related part of the Gnutella protocol before presenting our proposed DiCAS. Topology maintenance and search operations of the Gnutella network are specified in [18]. Each Gnutella peer connects to several overlay neighbors using point-to-point connections. A peer sends *ping* messages periodically to check all connections with its direct neighbors, and expects the *pong* messages from them. Typical Gnutella peers will try to maintain a pre-specified number (3 to 5 for a normal node, and much more for a ultrapeer) of connections. Gnutella peers overhear all the *pong* and Query Response messages passing by and cache IP addresses of other peers currently alive. If a peer detects that one of its neighbors is offline, it will look up its host cache or connect to a well-known Gnutella host cache sever, and randomly create another connection.

In order to locate a file, a source peer floods a query to all its direct neighbors. When a peer receives a query, it checks its local index to see whether it has the queried content. If so, a query response will be returned along the reverse of the query path to the source peer. Otherwise, the query will continue to be broadcasted. In the current Gnutella protocol, query responses are not cached by any peers in the returning path.

4.2. Distributed Caching

In addition to a *local index* that keeps indices of local files, each peer maintains a *response index* which caches the query results that flow through the peer. Each item cached in the *response index* includes the queried file name, and the IP address of the responding peer where the file is located. When a peer receives a query from its neighbor, it will look up the *response index* as well as the *local index*. A query match with either of them will generate a response. Instead of caching query responses in all peers along the returning path, *Distributed caching* attempts to cache the responses in some selected peers. The key of *distributed caching* is to determine whether an incoming query response should be cached or not so that the duplicated query responses among neighboring peers can be minimized. In DiCAS, when a peer joins the P2P system, it will randomly take an initial value in a certain range [0..M-1] as its group ID so that all the peers are separated into M groups. A uniform hash algorithm is employed to translate the queried file name string to a hash value. We define that a query matches a peer if the equation below can be satisfied.

$$Peer\ Group\ ID\ =\ hash(query)\ Mod\ M$$

For a passing query response, each peer overhearing the response independently performs a computation on the response using the hash function, and caches this response only when this hash value matches the peer's group ID. For example, when M = 2, all peers are separated into two different groups. Suppose the modulus operation result of a file name's hash value equals 1. Only the peers in group1 will cache this response, as illustrated in Figure 5.

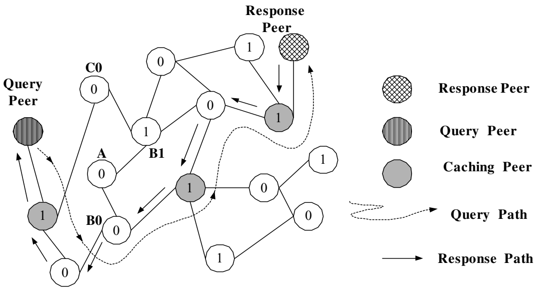

Figure 5 Cache strategy of DiCAS

4.3. Adaptive Search

Accordingly, a query is also forwarded to only neighbors with a group ID that matches the hash value of the desired file name in the query. For example, when a node receives a query which matches group ID of 1, the query will only be forwarded to neighbors with group ID of 1. We claim the group ID is uniformly distributed in the P2P network due to its value being randomly chosen. Benefiting from the group ID's uniform distribution, a query can be forwarded to matched neighbors in most cases. However, it is still possible that query forwarding can be blocked if none of a peer's neighbors have a matched group ID. To avoid the early death of the query, the peer will select a neighbor with the highest connectivity degree to forward the query to in this case. Based on the *adaptive search* algorithm above, the query forwarding will be restricted to peers with the matched group ID. Those peers form a virtual layer which has much smaller searching space than the original P2P network. Based on the modulus operation, the whole network is logically divided into multiple layers and each query will be forwarded within the correspondent layer with matched group ID.

5. Simulation Methodology

5.1. Considerations of P2P Simulation

It is unrealistic for us to make a considerable number of peers in Gnutella network configured with the support of DiCAS for the purpose of evaluating the performance improvement. We decided to develop a DiCAS simulator for a large-scale cache-aware P2P network. We choose to simulate each peer's message-level behaviors as an effort to investigate searching and index caching on all peers across the entire network. Each simulated peer is able to send queries, modify local and response index caches, and generate responses based on both caches. Our previous experiences on network simulations and experiments show that simulation configurations and parameters strongly influence the validity of simulation results. In this section, we summarize a list of network parameters used in the simulations of previous studies.

The parameters that determine the simulation scenarios fall into three categories: network and topology parameters, workload parameters, and initial content/keyword distributions over the network. Content popularity at a publisher follows Zipf-like distribution (aka Power Law) [19, 20], where the relative probability of a request for the *i*th most popular page is proportional to $1/i^{\alpha}$, with α typically taking on some value less than unity. The observed value of the exponent varies from trace to trace. The request distribution does not follow the strict Zipf's law (for which $\alpha=1$), but instead follows a more general Zipf-like distribution. Query word frequency does not follow a Zipf distribution [21, 22]. User's query lexicon size does not follow a Zipf distribution [21] but with a heavy tail.

Both the overall traffic and the traffic from the 10% most popular nodes are heavy-tailed in terms of host connectivity, traffic volume, and average bandwidth of the hosts [7]. Paper [23] suggests a log-quadratic distribution ($10^{-\alpha^2}$) for stored file locality and transfer file locality. The length of time that nodes remain available follows a log-quadratic curve [23], which could be approximated by two Zipf distributions.

Research on content searching in P2P networks generally uses simulation to illustrate the effectiveness of the underlying approach. Thus the problem of choosing a decent abstraction level becomes a critical issue, which in turn determines what simulation configuration is needed for such a scenario. For specific simulation, one should carefully choose related parameters and distributions such that the simulation results and observations are reasonable.

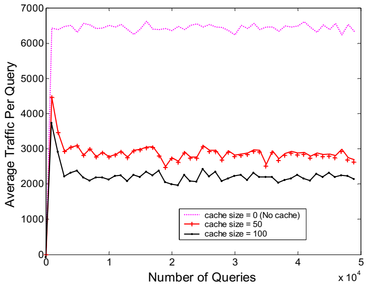

Figure 6 Average traffic incurred by each query UIC vs. No cache

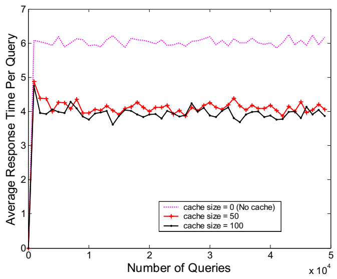

Figure 7 Average query response time UIC vs. No cache

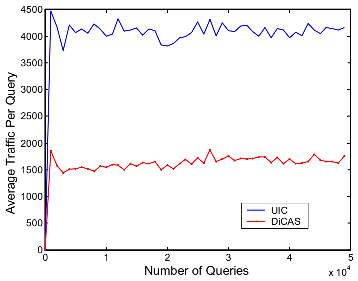

Figure 8 Average traffic incurred by each query

5.2. Our Simulation Configuration

In our simulation configuration, we generate two types of network topology: Power-law topology and random topology with an average connectivity degree of 3. We examine the impact of index caching on searching efficiency in terms of keyword matching. Hence in our simulation we only look at single keyword matching rather than document matching and semantic layer searching. Blind flooding in Gnutella network is simulated by conducting the Breath First Search algorithm from a specific node. A search operation, bounded by TTL of 7, is simulated by randomly choosing a peer as the sender, and a keyword according to Zipf distribution. In each simulation session, a large number of search operations are simulated sequentially. While receiving a query, a peer will consult its local index and its query response index cache using the searching keyword for possible matches. The trace we collected (described in Section 3.2) is used in our simulation.

6. Performance Evaluation

A well-designed search mechanism should seek to optimize both efficiency and user satisfaction. Efficiency focuses on better utilizing resources, such as bandwidth and processing power, while user satisfaction focuses on user-perceived qualities, such as number of returned results and response time. We will use three performance metrics: query success rate, query response time, and traffic overhead incurred by queries to evaluate the effectiveness of DiCAS.

6.1. Effectiveness of Uniform Index Caching

In the first simulation, we examine the effectiveness of *uniform index caching* (UIC) scheme in which all peers in a query response path will cache the query response. Blind flooding is still used in UIC to forward queries.

Figure 6 shows the average traffics incurred by each query, and Figure 7 shows the average query response times for different cache sizes. Not surprisingly, introducing caching query responses with a moderate cache size of 50 significantly reduces network traffic by 54%, and query response time by 33%. However, further increasing cache size in each peer would not improve performance proportionally. One of the reasons we have mentioned is that there exists a large amount of overlapped query responses among neighboring peers in UIC, which can limit the performance improvement of caching query responses.

6.2. Effectiveness of DiCAS

Aiming at further improving search efficiency, we propose DiCAS to cache query responses in selected peers and forward the query to peers with matched group ID. DiCAS is evaluated in this section using M=2, which logically divides the search space into two layers.

Figure 8 and Figure 9 compare the average traffic, and the average query response time of UIC and DiCAS, respectively. We can see that DiCAS outperforms UIC by 70% in terms of average traffic reduction. Compared with UIC, DiCAS increases average query response time by only about 6%.

When we measure the success rate of UIC and DiCAS, we find that UIC can keep the same query success rate as original flooding without caching (see Figure 10). However, Figure 11 shows that query success rate of DiCAS is decreased by 13% compared with UIC. Because the DiCAS protocol only forwards a query to some selected neighboring peers instead of all neighboring peers, it is likely that the query will miss some peers who have queried results. There are two reasons for a query to miss matched peers.

First, some matched peers may be missed. In DiCAS, a source peer forwards its query to those neighboring peers whose group ID matches the query. Some other neighbors are non-matched neighbors. However, the non-matched neighbors' neighbors may have matched group ID with this query, but may be never reached by the query. See Figure 5 again for an example, Peer *A* has two neighbors *B1* and *B0*. Peer *C0* is *B1*'s neighbor, but two hops away from *A*. Assume that peer *B0* and *C0* have the same group ID (e.g. *GID=0*), and peer *B1* has another group ID (e.g. *GID=1*). If peer *A* initiates a query that matches *GID=0*,

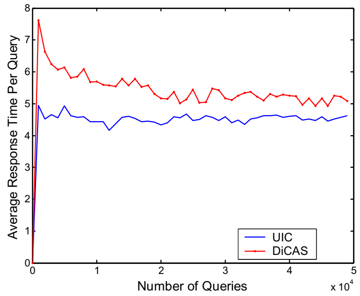

Figure 9 Average query response time

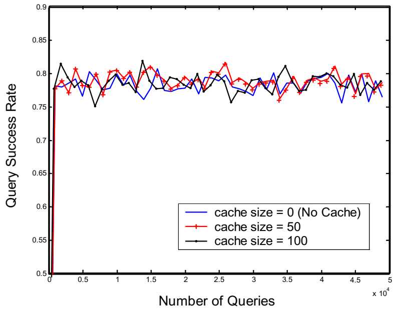

Figure 10 Average success rate of each query UIC vs. No cache

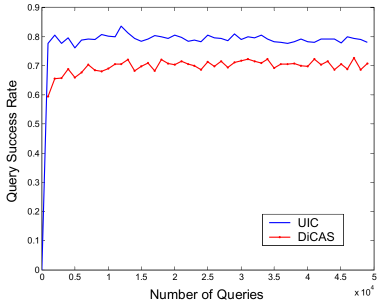

Figure 11 Average success rate of each query

the query will only be forwarded to peer *B0*. Peer *B1* will not receive the query, so the query may not reach *C0*, but *C0* is indeed a matched peer should be queried.

Second, some matched objects may be missed. When a peer joins, it selects a group ID, but this cannot guarantee that all its local objects will match the group ID. In this case, there are some objects that do not match the owner's group ID and will never be queried, forming some dead corners. Thus, some of the objects, even though they are available, may not be found by many queries.

Motivated by above two reasons, we proposed two solutions to address the problem of query success rate degradation in DiCAS, which are described and evaluated next.

6.3. Solutions to improve the query success rate

The first solution is called *push-DiCAS* that attempts to avoid missing matched objects. When a peer is joining a P2P network and randomly taking a group ID, it computes hash values of the file names for all its sharing objects. If some objects do not match this peer's group ID, the peer will push the indices of these objects to one of its neighboring peers with matched group ID. These neighboring peers will cache the indices of pushed objects with a similar format of a query response indicating whereabouts of the objects. If none of the neighboring peers with matched group ID exists, a peer with the highest connectivity degree will be selected. The whole process is repeated until a peer with matched group ID is found. Figure 12 shows the average query success rates of UIC and *push-DiCAS*. We can see that the query success rate of *push-DiCAS* is improved to be very close to that of UIC. Figure 14 shows average traffic comparisons. The push message travels along a single path. Thus, compared with the exponentially increasing query flooding, the increased traffic caused by the *push* operation is trivial, which is shown by the difference between the *push-DiCAS* curve and *DiCAS* curve.

We called the second solution *random-DiCAS* that attempts to avoid missing matched objects shared by non-matched peers. Instead of ignoring all non-matched neighboring peers, a peer forwards its query to some randomly selected non-matched neighboring peers, but these non-matched peers will not forward the query further. As a result, the flooding is mainly restricted within matched peers while some of the non-matched peers are still covered. Figures 12-14 show *random-DiCAS*'s comparable performance with *push*-DiCAS in average query success rate and average traffic.

Comparing *push-DiCAS* and *random-DiCAS*, we find that they have comparable performance except that random-DiCAS causes a little bit more traffic because more neighbors will be queried on average. The key factor affecting the performance of *push-DiCAS* is the frequency of the push operation, which balances the volume of search traffic and success rate. The push frequency heavily depends on the dynamic nature of P2P network. Because peers can join and leave at any time, it is possible that peers who have received pushed indices leave the network. In these cases, the source peer should do *push* operations frequently, which incurs extra traffic. However, compared with the exponentially increased search traffic, the linearly increased push traffic is trivial. Figure 14 shows that the overall traffic is still reduced significantly. In our simulation, we investigate the case of M=2 thoroughly , which means that the search space is divided into two layers. When the network is divided into more layers, the volume of flooding traffic can be further reduced. However, the push messaging between multiple layers will become heavier. The best choice to balance well the flooding traffic and push overhead depends on the dynamic nature of the real Gnutella network, which will be studied in our future work.

7. Conclusion

The DiCAS protocol, which distributes index cache among peers and divides the searching space into multiple layers, can significantly reduce the searching traffic in Gnutella-like P2P network. Our simulation results demonstrate its strong effectiveness under different conditions. We

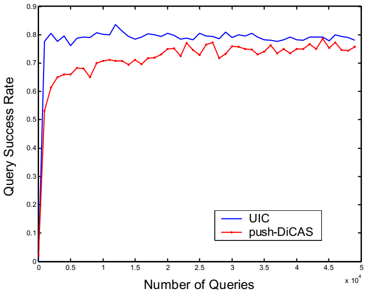

Figure 12 Average success rate of each query

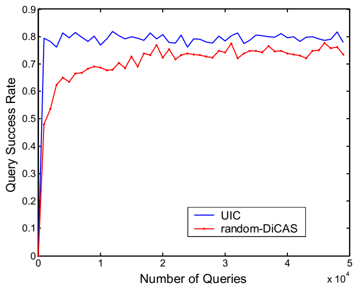

Figure 13 Average success rate of each query

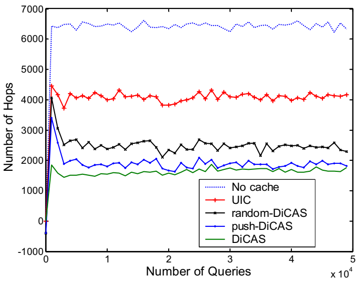

Figure 14 Average traffic incurred by each query

have also shown that deploying such a caching scheme in an existing P2P network, such as Gnutella, is feasible with an immediate favorable impact on P2P search performance, thus making unstructured P2P systems more scalable. We are refining a prototype version of the Gnutella-based DiCAS for public release in the P2P community.

References

[1] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," *in Proceedings of SIGCOMM*, 2001.

[2] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," *in Proceedings of International Conference on Distributed Systems Platforms*, 2001.

[3] B.Y.Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-resilient wide-area location and routing," Technical Report UCB//CSD-01-1141, U.C.Berkeley 2001.

[4] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," *in Proceedings of ACM SIGCOMM*, 2001.

[5] Gnutella, http://gnutella.wego.com/

[6] M. Ripeanu, A. Iamnitchi, and I. Foster, "Mapping the Gnutella Network," *IEEE Internet Computing*, 2002.

[7] S. Sen and J. Wang, "Analyzing peer-to-peer traffic across large networks," *in Proceedings of ACM SIGCOMM Internet Measurement Workshop*, 2002.

[8] S. Saroiu, K. P.Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy, "An Analysis of Internet Content Delivery Systems," *in Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, 2002.

[9] Fasttrack, http://www.fasttrack.nu

[10] Why Gnutella can't scale. No, really, http://www.tch.org/gnutella.html

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

[12] Y. Liu, X. Liu, L. Xiao, L. M. Ni, and X. Zhang, "Location-Aware Topology Matching in Unstructured P2P Systems," *in Proceedings of INFOCOM 2004*, 2004.

[13] B. F. Cooper and H. Garcia-Molina, "Studying search networks with SIL," *in Proceedings of IPTPS*, 2003.

[14] The popularity of Gnutella queries and its implications on scalability, http://www2.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html

[15] E. P. Markatos, "Tracing a large-scale peer to peer system: an hour in the life of gnutella," *in Proceedings of the 2nd IEEE/ACM International Symp. on Cluster Computing and the Grid 2002*, 2002.

[16] S. Patro and Y. C. Hu, "Transparent Query Caching in Peer-to-Peer Overlay Networks," *in Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS)*, 2003.

[17] Limewire, http://www.limewire.com

[18] The Gnutella protocol specification 0.6, http://rfc-gnutella.sourceforge.net

[19] V. Almeida, A. Bestavros, M. Crovella, and A. d. Olivera, "Characterizing Reference Locality in the WWW," *in Proceedings of the IEEE Conference on Parallel and Distributed Information Systems (PDIS)*, 1996.

[20] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications," *in Proceedings of INFOCOM'99*, 1999.

[21] Y. Xie and D. O'Hallaron, "Locality in Search Engine Queries and Its Implications for Caching," *in Proceedings of INFOCOM('02)*, 2002.

[22] B. J. Jansen, A. Spink, J. Bateman, and T. Saracevic, "Real Life Information Retrieval: a study of User Queries on the Web," *SIGIR Forum*, vol. 32, pp. 5-17, 1998.

[23] M. T. Schlosser and S. D. Kamvar, "Availability and locality measurements of peer-to-peer file systems," *in Proceedings of ITCom: Scalability and Traffic Control in IP Networks*, 2002.