Exploiting Content Localities for Efficient Search in P2P Systems*

Lei Guo[1], Song Jiang[2], Li Xiao[3], and Xiaodong Zhang[1]

[1] College of William and Mary, Williamsburg, VA 23187, USA
{lguo,zhang}@cs.wm.edu
[2] Los Alamos National Laboratory, NM, 87545, USA
sjiang@lanl.gov
[3] Michigan State University, East Lansing, MI 48824, USA
lxiao@cse.msu.edu

Abstract. Existing P2P search algorithms generally target either the performance objective of improving search quality from a client's perspective, or the objective of reducing search cost from an Internet management perspective. We believe that the essential issue to be considered for designing and optimizing search algorithms in unstructured P2P networks is the trade-off between the two performance objectives. Motivated by our observations, the locality of content serving in the peer community and the localities of search interests of individual peers, we propose *CAC-SPIRP*, a fast and low cost P2P searching algorithm. Our algorithm consists of two components. The first component aims to reduce the search cost by constructing a *CAC* (**C**ontent **A**bundant **C**luster), where content-abundant peers self-identify, and self-organize themselves into an inter-connected cluster providing a pool of popular objects to be frequently accessed by the peer community. A query will be first routed to the CAC, and most likely to be satisfied there, significantly reducing the amount of network traffic and the search scope. The second component in our algorithm is client oriented and aims to improve the quality of P2P search, called *SPIRP* (Selectively Prefetching Indices from Responding Peers). A client individually identifies a small group of peers who have the same interests as itself to prefetch their entire file indices of the related interests, minimizing unnecessary outgoing queries and significantly reducing query response time. Building SPIRP on the CAC Internet infrastructure, our algorithm combines both merits of the two components and balances the trade-off between the two performance objectives. Our trace-driven simulations show that CAC-SPIRP significantly improves the overall performance from both client's perspective and Internet management perspective.

1 Introduction

The effectiveness of content search in unstructured P2P networks such as Gnutella [6] and Kazaa [8] can be measured by two performance objectives that

* This work is supported in part by the U.S. National Science Foundation under grants CNS-0098055 and CCF-0129883.

R. Guerraoui (Ed.): DISC 2004, LNCS 3274, pp. 349–364, 2004.

may have conflicting interests. The first objective coming from each *individual peer*'s perspective is to improve its *search quality*, i.e., to increase the number of effective results and to minimize the response time of each query. The second objective coming from the Internet management perspective is to reduce the total *search cost* of the *peer community* (all peers in the system), i.e., to minimize the network bandwidth consumptions and other related overheads, such as CPU and storage demands. Existing search algorithms generally aim at one of the objectives and have performance limits on the other. For example, flooding search targets to maximize the number of search results for each peer but results in too much traffic in the system, while random walking [11] target to reduce the search traffic for the system but can lead to long response time for individual peers.

We believe that the essential issue of designing and optimizing search algorithms in unstructured P2P networks is the trade-off between the two performance objectives. In this paper, we analyze the content serving regularities in the peer community and the search patterns of individual peers, and show there exist two kinds of localities in P2P content search. (1) The *locality of content serving* in the peer community: most search results are served by a small number of content-abundant peers. (2) The *localities of search interests* of individual peers: peers generally target contents on a few topics of interests, and can get most requested objects from a small number of peers with the same interests as themselves. Motivated by these two observations and the trade-off between the two performance objectives, we propose *CAC-SPIRP*, a fast and low cost P2P searching algorithm.

CAC-SPIRP algorithm comprises two complementary techniques, *CAC* and *SPIRP*. CAC technique aims to reduce the search cost by exploiting the content serving locality among the peer community. In this technique, a small number of content-abundant peers are self-identified based on their query-answering histories, and self-organized into a cluster called *CAC* (**C**ontent **A**bundant **C**luster), which serves as a pool of popular objects to be frequently requested. SPIRP technique is client oriented. By **S**electively **P**refetching **I**ndices from **R**esponding **P**eers, the search interest localities of individual peers can be well exploited to speedup query processing. By combining both techniques, CAC-SPIRP algorithm is highly effective in addressing the trade-off between the two performance objectives, and does not produce additional overheads in P2P networks, where the content-abundant cluster is constructed, and the SPIRP technique is facilitated in each peer, retaining the merits of both CAC and SPIRP.

Current unstructured P2P systems have been evolved from a Gnutella-like structure to a super-node [10] based system where high bandwidth peers serve as super-nodes to maintain the indices of their leaf nodes, such as Morpheus [12] and Kazaa [8]. Although the super-node structure has effectively reduced the overall network bandwidth usage of P2P search, it does not consider the trade-off between *search quality* and *search cost*. We have the following three reasons why our CAC-SPRIP solution effectively addresses the three major limits of super-node based systems.

1. *Index based super-nodes limit search ability to certain applications.* Since the index service is file name based, the search scope can be seriously limited in comparison with a full-text content-based search. Another example of the super-node limit is related to the content search for encrypted files. When a peer searches the content of an encrypted file (or a media file with a private decoder, such as DVD video), a secrete key (or a specific codec) in the query is required, however, an index server will not be able to support this kind of search. In contrast, our CAC-SPIRP is content-based, where the content abundant cluster directly provides search service to peers, thus can support a generic search in P2P systems.
2. *A super-node server is not responsible for the content quality of its leaf nodes.* Thus, a super-node service does not guarantee the search quality. In contrast, our content abundant cluster with a prefetching support aims at providing high quality search results to peers.
3. *Super-node structure is becoming inefficient.* In order to address the potential single-point-failure problem, in existing super-node based P2P systems, a peer has to connect to multiple super-nodes, which creates an increasingly large number of super-nodes on the Internet, significantly adding the maintenance costs of super-nodes. In contrast, our content abundant cluster is highly resilient by hosting a large number of powerful peers, where the single point of failure problem does not exist.

The remainder of the paper is organized as follows. Section 2 discusses some existing P2P searching algorithms. Section 3 presents our measurement observations. Sections 4 and 5 present the two searching techniques we proposed, CAC and SPIRP, respectively. Section 6 describes the CAC-SPIRP algorithm. Section 7 evaluates the two techniques and our proposed algorithm. We summarize our work in Section 8.

2 Related Work

Other solutions have been proposed for efficient search in unstructured P2P systems. *Directed BFS* [15] attempts to take advantage of the irregular content distribution by using very limited local information of the search history instead of fully exploiting the skewness of content distributions like CAC. *Random walk* search such as [11] and [3] can reduce search traffic effectively but can only give a small number of results and have long response time. *LightFlood* [7] aims to reduce the redundant messages of broadcast operations in P2P systems. *Interest-based locality* approach [14] shares the similar principle of SPIRP by exploiting the common interests among different peers. However, in this approach, a requesting peer connects to a small number of peers with same interests directly, limiting the locality of interests that can be exploited. Further more, the time for building interest groups is non-trivial, and these interest groups are not useful when the peer offline and online again, due to the transient nature of P2P networks. In contrast, our CAC scheme provides a persistent public service for all peers in the system.

3 Characterizing the Localities in the Peer Community and Individual Peers

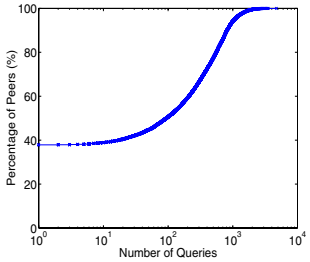
Existing measurement studies such as [1] and [13] investigated the file distributions in P2P systems and show a small percentage of peers share much more number of files than other peers in P2P systems. Study [4] investigated the locality of files stored and transferred in P2P systems and found that a small percentage of popular files account for most shared storage and transmissions in P2P systems. However, a peer with many files does not necessarily mean that it can provide corresponding content service: some of files may be never accessed, or accessed rarely. In this section, we present our experimental observations on the P2P search patterns and content distributions, and characterize the content serving locality in the peer community and the search interest localities of individual peers.

3.1 Data Preparation

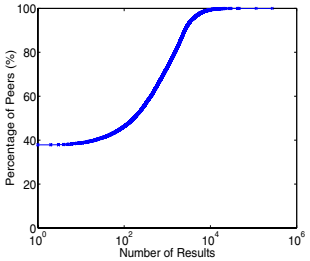
We built Gnutella crawlers based on the open source code of LimeWire Gnutella [9] and conducted a 4-day crawling to collect traces. We monitored the query sending of 25,764 different peers during their life time, and collected 409,129 queries. We randomly selected 1,600 peers and their corresponding queries (total 25,093) as the *P2P client set* for our study. We also collected the entire indices of 18,255 different peers and estimated that there are 37% free-riders in Gnutella networks. We used all index traces as well as the corresponding free-riders as the *P2P server set* (total 29,050 different peers) for our study. Finally, we matched all queries sent by peers in the P2P client set with all indices of peers in the P2P server set to complete the data preparation.

3.2 The Locality of Content Serving in the Peer Community

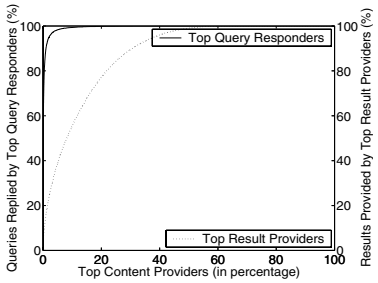
In our study, we ranked all peers in the P2P server set by the total number of queries they can reply and by the total number of results they can provide respectively, since a peer can reply a query with multiple results. Figure 1(a) shows the distribution of the number of queries that peers can respond. We can see a significant heterogeneity of the ability to reply queries among Gnutella peers: there are only about 6% peers that can reply more than 1,000 queries (4% of all queries) each, while there are more than 50% peers that can only reply less than 100 queries (0.4% of all queries) each. Figure 1(b) shows the distribution of the number of results that peers can provide. We can see that similar heterogeneity exists in this case as well: there are only about 10% peers that can provide more than 2,500 results (0.1 results per query on average) each, while there are more than 60% of peers that can only provide less than 500 results (0.02 results per query on average) each. We call those peers who can reply significantly more queries than other peers as *top query responders*, and call those peers who can provide significantly more results than other peers



(a) The CDF of queries that peers can reply.



(b) The CDF of results that peers can provide.



(c) The cumulative contributions of top content providers.

Fig. 1. The skewness of Gnutella peers' abilities to reply queries and to provide results.

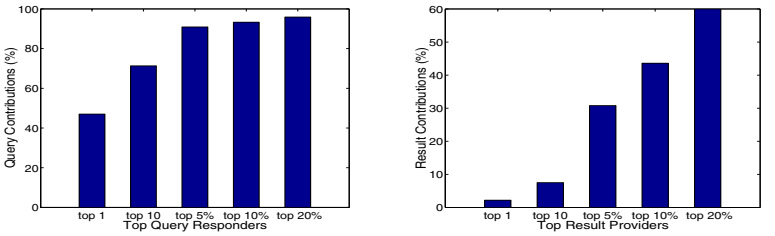
as *top result providers*. We observed in most cases that a top query responder of the peer community is also a top result provider of the peer community, and vice versa. For example, 84% of peers in the top 10% query responder set are in the top 10% result provider set as well. Therefore, we call both of them as *top content providers*.

Next we study the cumulative contribution of these top content providers. We computed the union set of queries replied by top query responders and the cumulative number of results provided by top result providers, shown in Figure 1(c). We can see that the top 5% query responders can reply more than 98% of all queries altogether while the top 10% result providers can provide about 55% of all results in the system altogether.

The experiments above show strong *locality of content serving* in the peer community: a small percentage of peers (top content providers) account for most content contributions in the system.

3.3 The Localities of Search Interests of Individual Peers

The access patterns of individual peers differ from that of the peer community as a whole. In the following experiments, we try to get insight of the search behaviors of individual peers. We only consider queries that have been replied by other peers, and simply call them *replied queries*. We selected peers having at least 10 replied queries in the P2P client set as *requesting peers*. For a requesting peer, we



(a) The average contributions of top query responders.

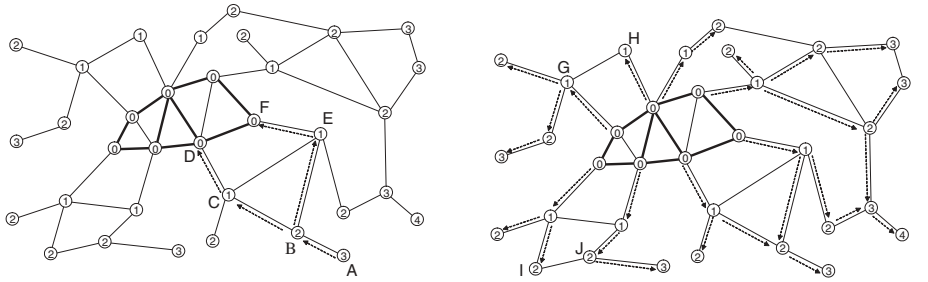
(b) The average contributions of top result providers.

Fig. 2. The average contributions (in percentage) of top query responders and top result providers for peers having at least 10 replied queries.

define the *query contribution* of its each responder as the number of queries the responder has replied, and define the *result contribution* of its each responder as the number of results it has provided. We ranked each peer's responders by their query contributions and by their result contributions respectively.

Figure 2 shows the average query contributions of requesting peers' top query responders and the average result contributions of requesting peers' top result responders. The contributions are normalized by the overall contributions of all responders of the corresponding requesting peers. In Figure 2(a), the 5 bars represent the average contributions of the top 1, top 10, top 5%, top 10%, and top 20% query responders of requesting peers, respectively. The top 1 query responder of a requesting peer is a single peer who responds the highest number of queries. This responder can respond 47% of all replied queries on average. The top 5% query responders together can respond about 91% of all replied queries. Figure 2(b) shows that the top 10% result responders of the requesting peers account for about 31% of all results they receive on average, and that the top 20% result responders account for about 60% of all results on average. Figure 2(a) and 2(b) also show the top 10 query responders of requesting peers can reply 71% of all replied queries on average, and the top 10 result responders of requesting peers can provide about 7.5% of all results on average. Further studies show that the top content providers of individual requesting peers are their top query responders, because a peer answering a query with many results is not necessarily able to answer other queries. Our studies also show that the top query/result providers of an individual peer are not necessarily the top content providers of the peer community, because the former depends on the search interests of the requesting peer, while the latter depends on the group behaviors of the whole community.

The experiments above show a small number of top query responders of individual requesting peers account for most content contributions of these peers. This fact indicates that the requesting peers and their top query responders have the same interests in content searching and content sharing respectively. From the aspect of clients, there exist strong *localities of search interests* for individual peers: a peer's requests generally focus on a few interest topics, and it can be satisfied by a small number of peers with the same interests.



(a) Up-flowing (b) Down-flooding

Fig. 3. The up-flowing and down-flooding operations. Each circle denotes a peer in the P2P network, and the number in the circle is the level of this peer. The bold lines denote the CAC links and the thin lines denote the original P2P links. (a) shows the routing paths of a query sent by peer A at level 3. The query is routed along the dash lines, passing through B, C, D and B, E, F until reaching the CAC. A query may have multiple paths to reach the CAC, improving the robustness of query routing. (b) shows the routing paths of down-flooding. The query is concurrently routed from all CAC peers to level 1 peers, then level 2 peers, until reaching peers with the maximal level. Only links from lower level peers to higher level peers are used to route messages; links such as GH and IJ are not used for query routing.

4 Constructing Content-Abundant Cluster (CAC)

The basic idea of CAC technique is to have a number of content-abundant peers in the peer community self-organized into a *content-abundant cluster* (CAC) to actively serve contents for the entire system. By being directed into the cluster for an efficient search first, most queries can be satisfactorily answered without meaninglessly bothering those content-scarce peers. Because these content-scarce peers account for a significant portion of the peer community, a large amount of network traffic and computational cost can be saved. For a small number of queries that can not be satisfied in the cluster, we relay them out of the cluster in an efficient fashion. The key components of the CAC technique are presented as follows.

System Structure. The content-abundant peers are those top content providers of the peer community. In CAC technique, we allow peers self-evaluate the quality of content service they can provide based on the history of their query-answering. The criterion of *content service quality* is the percentage of queries a peer can reply. Peers whose content service qualities reach a threshold are CAC member candidates and have the same possibility to join the CAC. Both the quality threshold for CAC members and the CAC size can be predefined or updated periodically by some mechanism to adapt to the dynamics in P2P systems. Our simulations show that CAC has no strict requirements on these two parameters (see Section 7.3).

CAC technique is modular and can be easily deployed on top of any P2P overlays. The content-abundant cluster is a connected overlay independent of the

original P2P overlay. There are two types of links in P2P systems implementing CAC technique: one is the original P2P overlay links, the other is the CAC overlay links. Each peer in the system is assigned a *level*: the level of each CAC peer is defined as 0, and the level of an non-CAC peer is defined as the number of hops from this peer to the nearest CAC peer. When the CAC overlay or the P2P overlay changes, the level values of relevant peers can be updated one by one quickly. By using levels, the unstructured P2P system is organized logically for efficient and robust query routing without changing the original P2P overlay.

Query Routing. A query is routed from higher level peers to lower level peers until reaching the CAC, shown in Figure 3(a). We call this operation *up-flowing*. As soon as a query enters the CAC, it is flooded in the CAC to search contents.

The responses of a query are routed back to the requester along the same path that it comes. The requester waits a period of time for the arrival of responses from the CAC, called the *response waiting time*. If the requester does not get enough number of results during the waiting time, the query will be routed in the entire system for a global search as follows. First, the query is up-flowed to and then flooded in the CAC again. Upon receiving the query, each CAC peer propagates it to level 1 peers immediately. Then the query is propagated from lower level peers to higher level peers in the P2P overlay. We call this operation *down-flooding*, shown in Figure 3(b). Down-flooding is much more efficient than simply flooding the query in the P2P overlay because only links between two successive levels of peers are used for propagating queries, reducing a great amount of unnecessary traffic.

Different from having each non-CAC peer directly connect to a CAC peer like super-node approach, CAC technique is more robust in query routing, because the failures of individual CAC peers have little effects on those non-CAC peers.

System Maintenance. The content-abundant cluster is maintained in a proper size. Each CAC peer holds the value of the CAC size locally and updates the value periodically by broadcasting ping messages and receiving corresponding pong messages in the CAC.

Each self-identified content-abundant peer, p, tries to join the CAC by up-flowing *join* requests periodically until success. Upon receiving a join request message, a CAC peer, P, accepts or denies the request based on the CAC size value it holds. If P believes the CAC needs more members, it accepts the request and sends a list of randomly chosen CAC members back to p. Then p randomly selects several of them as its neighbors to join the CAC. These CAC members can still reject the connection request based on their local values of the CAC size, preventing malicious attacks such as adding members to the CAC repeatedly.

CAC peers who can not provide qualified services for some period of time and overloaded peers leave the CAC to become a normal peer. Before leaving the CAC, a peer broadcasts a *leave* message to let other CAC peers update the CAC size values they hold. Even if a CAC peer disconnects abnormally, other CAC peers can still update the size values when broadcasting the next ping.

5 Selectively Prefetching Indices from Responding Peers

SPIRP technique is client oriented and motivated by the search interest localities of individual peers. Although the contents in a typical P2P network are huge and highly diverse, each peer's interests are limited and generally focused on a few topics. Queries from a requesting peer can be frequently answered by a small number of serving peers, as Section 3.3 shows. In SPIRP, after receiving answers to its initial queries, a client selectively identifies a small group of responders who have the same interests as itself, and asks them to send their entire file indices of the related interests to this client. With SPIRP, the number of outgoing queries is minimized in the client side by exploiting the common interests between the requesting peer and its responders, reducing both the response time and bandwidth consumptions. In addition, since each requesting peer only prefetches and maintains the file indices of a limited number of peers, the index transmission overheads and the storage requirement are small.

5.1 Data Structure

The basic data structure of SPIRP in each peer consists of several key components. Each peer maintains a set of indices of files to be shared in the P2P network, called the *outgoing index set*. It also maintains a set of indices selectively prefetched from its responders, called the *incoming index set*. In addition, each peer maintains a set of responders who have replied to it, called the *responder set*. This set is organized as a hash table in which the key is the responder's GUID (Globally Unique Identifier) and the value is the responder's meta data. The major fields of a responder's meta data are shown in Table 1. The responder set is also ranked as a *priority queue*, where the *priority* is defined as the number of queries a responder has responded so far.

Table 1. The data structure of responder's meta data.

Field	IP addr.	port	is cached	priority	index size	timestamp	expire time	update time
Bytes	4	2	2	4	4	4	4	4

A peer does not keep any information for other peers prefetching its indices. To help these peers refresh the indices they prefetched, each responding peer averages its previous on-line session durations as the estimated duration of its current session, and averages its previous update intervals as the estimated update interval. When indices in a responding peer are prefetched, the estimated expire time, update time, and current timestamp of the peer are piggybacked to the requesting peer.

5.2 SPIRP Operations

With the support of the above data structure, several key SPIRP operations are defined as follows.

Sending Queries. Initially the incoming index set and the responder set are both empty. As a requesting peer sends a query, it searches the incoming index set first. If any indices match the query, the requesting peer checks if the corresponding responders are still alive (see the *Checking Index Expiration* operation) and then returns the available matched results to the user. If the query cannot be satisfied locally, the peer sends the query to the P2P network in a normal way (e.g., flooding search), and then returns the corresponding results to the user. Then the peer updates the responder set and the priority queue.

Prefetching and Replacing Indices. The requesting peer asks those high priority responders, which are not in the incoming index set currently, to send their related indices until the incoming index set is full. When the priority queue changes, a simple replacement policy is used. The peer removes the indices of low priority responders from the incoming index set and prefetches the indices of high priority responders that are not in the incoming index set currently.

Checking Index Expiration. When the estimated expiration time of a responder reaches, or a query hits its incoming index set, the peer checks if the responder is still alive. If not, the peer deletes its indices from the incoming index set and its meta data from the responder set, then updates the priority queue.

Checking Index Update. When the estimated update time of a responder reaches, the peer sends the responder the timestamp of the prefetched indices to check if update happens. If yes, the responder sends the difference set of the indices or simply resends the whole index set.

6 CAC-SPIRP: Combining CAC and SPIRP Techniques

The CAC technique has its strong merits on reducing both bandwidth consumption and client response time when the requests success in the CAC, while the SPIRP technique shares the same advantage when the search interests is well exploited by the selective prefetching. However, each technique has its limits. Although the percentage of requests that fail in the CAC is small, the miss penalty can be non-trivial, negatively affecting the average latency. On the other hand, the flooding operations of outgoing queries in SPIRP produce a great amount of traffic. The motivation of CAC-SPIRP algorithm is to combine the merits of these two complementary techniques and tune the trade-off between the two performance objectives to improve the overall performance of P2P search.

SPIRP is client-oriented and overlay independent. On the other hand, CAC is an application-level infrastructure for unstructured P2P systems. Applying SPIRP technique on the CAC infrastructure, we have the CAC-SPIRP algorithm. The algorithm is simply to combine both CAC and SPIRP: the peers use SPIRP to prefetch file indices, and use CAC to route outgoing queries.

7 Experiments and Performance Evaluation

7.1 Simulation Methodology

In P2P systems, peers join and leave P2P network from time to time. A measurement study in Gnutella and Napster presented in [13] shows that the session

duration of peers follows heavy tail distribution, where the duration median is about 60 minutes. This study is consistent with our observations about the connection durations between the query collection crawler and Gnutella peers in Section 3.1. Study [2] further shows the lifespan of peers follows the Pareto distribution. Different from the simulations of existing studies such as [14], we considered the population dynamics in our evaluation, since the performance of SPIRP can be affected by the lifespan of responders. We assigned each peer in the P2P server set a random value of session duration following the Pareto distribution $P(x) = 14.5311 * x^{-1.8598}$ based on the statistics in [13] and our trace. We use the topology traces provided by Clip2 Distributed Search Solutions [5] and University of Chicago in our simulations. Due to page limit, we only present the simulation on a Gnutella snapshot of 6,946 nodes. Experiments on other topologies have similar results.

7.2 Performance Metrics

In P2P systems, the satisfaction of a content search depends on the number of results the user needs (10-50 results are in the normal range covering both low and high ends). In our simulations, we choose 1, 10, and 50 as the criteria of *query satisfaction* to show the search performances under different user requirements.

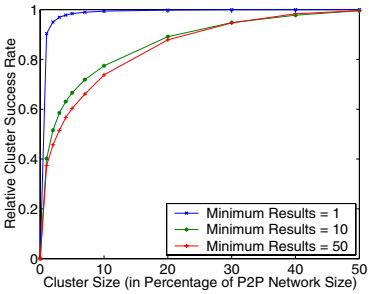
The metrics we used for content search in P2P systems are the *overall network traffic* in the system, the *average response time per query*, and the *query success rate*. The overall network traffic is a major concern of system designers and administrators, while the average response time and query success rate are major concerns of end users.

The overall traffic in our simulation comes from the accumulated communications of all queries, responses, and indices transferred in the network. Instead of modeling the actual network latency, we use the number of hops to measure the response time. The response time of a single result is measured by the number of a round trip hops from the requester to the responder, plus the response waiting time for CAC technique when the responder is not in CAC. For SPIRP technique, the response time of a result found in local index set is zero. The response time of a successful query is defined as the average response time of the first N results the requester receives, where N is the query satisfaction.

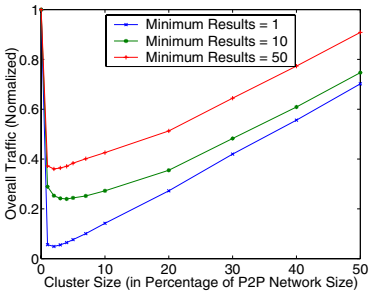
Both the flooding search and our CAC/SPIRP techniques can cover almost all peers in the system if necessary. What we are really concerned is not the absolute success rate of queries, but the *cluster relative success rate* for CAC technique, which is defined as the number of queries that can be satisfied in the cluster over the number of queries that can be satisfied by flooding search, and the *local relative success rate* for SPIRP technique, which is defined as the number of queries that can be satisfied in the incoming index set over the number of queries that can be satisfied by flooding search, respectively.

7.3 Performance Evaluation

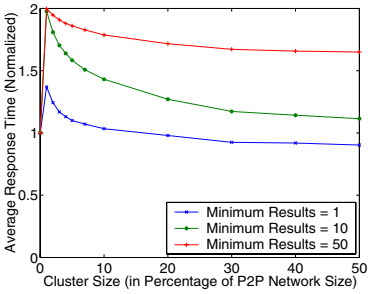
In this section, we first evaluate the effectiveness of CAC infrastructure, and then present the performance of CAC-SPIRP algorithm. We do not present the



(a) The cluster relative success rate.



(b) The overall traffic in the P2P network.



(c) The average response time per query.

Fig. 4. The performance of CAC technique under different query satisfactions and different sizes of clusters in which the cluster peers are those top query responders. The overall traffic and average response time are both normalized by the corresponding values of flooding search.

performance of SPIRP algorithm separately due to page limits. We chose the 1,600 peers in the P2P client set as requesting peers, and randomly placed these requesting peers on the simulated P2P network. Each requesting peer sends queries according to the corresponding timestamps in the query records.

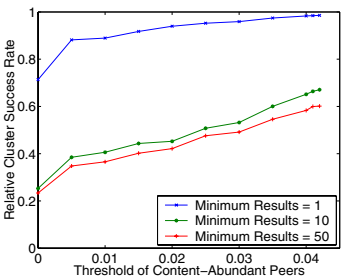
Performance Evaluation of CAC Technique. The effectiveness of CAC technique depends on both the cluster size and the capacities of cluster peers. Our first experiment evaluated the performance of CAC with different sizes of clusters to find a good cluster size. We chose the "best" content-abundant peers, those top N query responders, where N is the cluster size, as the cluster peers. We set the response waiting time as 12 hops. Changing the size of cluster, we have measured the cluster relative success rate, the overall network traffic, and the average response time per query for different query satisfactions.

Figure 4(a) shows the cluster relative success rates in clusters of different sizes for different query satisfactions. The cluster relative success rate increases with the increase of the cluster size, and decreases as the query satisfaction value increases. However, the curves of cluster relative success rates under 10 and 50 query satisfactions are quite close, indicating a high quality of content service of CAC. The cluster relative success rates are more than 55% for the cluster consisting of top 5% content providers, and more than 70% for the cluster consisting of top 10% content providers.

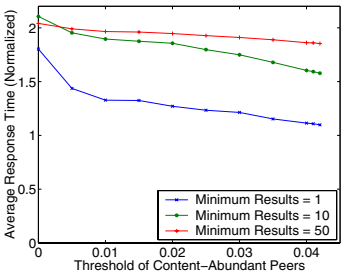
Although a large cluster helps to increase the cluster success rate, it increases the intra-cluster traffic as well. Figure 4(b) shows the overall traffic of CAC technique. We can see the cluster of top 5% content providers is effective enough in traffic reduction for all query satisfactions from 1 to 50. For example, compared with flooding search, CAC technique under this condition can reduce more than 90% traffic for queries that need only one result, more than 75% traffic for queries that need 10 results, and more than 60% traffic for queries that need 50 results.

The response time of CAC technique is not so good. Figure 4(c) shows the response time under different cluster sizes and query satisfactions. We can see the response time is higher than that of flooding algorithm unless the cluster size is very large and the query satisfaction is very small. This is because the flooding search can always find the shortest and fastest paths in the P2P network, while in CAC technique, both flooding in the cluster and up-flowing to the cluster consume time, and the response waiting time is a big penalty for queries not satisfied in the cluster.

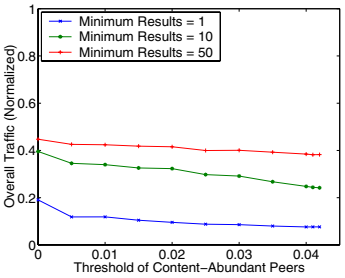
CAC technique randomly selects cluster peers from content-abundant peers instead of ranking them and selecting the best ones, which is not realistic in practice. Our second experiment evaluated the performances of CAC with different qualities of cluster peers in order to find a proper threshold for content-abundant peers. We set the cluster size as 5% of the community population size, measured the cluster relative success rate, overall traffic, and average response time under different thresholds for content-abundant peers. The results are presented as follows.



(a) The cluster relative success rate.



(b) The overall traffic in the P2P network.



(c) The average response time per query.

Fig. 5. The performance of CAC technique under different query satisfactions and different quality thresholds of content-abundant peers. The cluster size is set to 5% of the P2P network size. The overall traffic and average response time are both normalized by the corresponding values of flooding search.

Figure 5 shows that a high quality threshold of content-abundant peers helps to improve all performance metrics. However, the overall network traffic is not sensitive to the quality threshold, and the traffic can still be significantly reduced even the quality threshold is set to 0 (meaning the cluster peers are randomly selected from the peer community) due to the high efficiency of down-flooding. In the following experiments of this paper, we chose the threshold of content-abundant peers as 0.035, corresponding to peers who can respond 3.5% of all queries it receives. Under such a threshold, the overall traffic and the average response time are only 1.10 and 1.06 times of the corresponding performances of the ideal CAC systems. Meanwhile, the number of cluster peer candidates is about 1.7 times of the cluster size, indicating moderate population dynamics can not affect the system performance seriously.

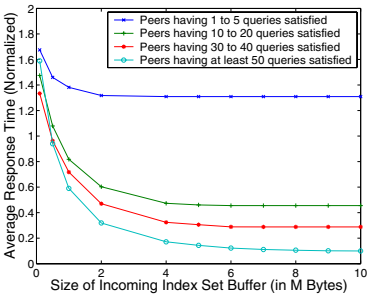
We have also compared the performance of CAC with iterative deepening [15] and expanding ring [11] algorithms. Our experiments show that the overall traffic of CAC technique is less than half of both algorithms, and the response time of CAC is lower than that of both algorithms under the same conditions. We do not present the figures due to page limit.

Performance Evaluation of CAC-SPIRP. CAC significantly reduces the overall network traffic at the expense of performance degradation in response time. SPIRP aims to reduce the response time though it can also decrease some network traffic as well. Both techniques only target one performance objective either from the perspective of system management or from the aspect of user experience. Our CAC-SPIRP algorithm considers the trade-off between these two performance objectives. Under certain conditions, the performance of CAC-SPIRP is nearly as good as that of SPIRP in terms of average response time reduction, and outperforms CAC in terms of the overall traffic reduction. The average response time, local relative success rate in the incoming index set buffer, and overall traffic of CAC-SPIRP algorithm are presented in Figure 6.

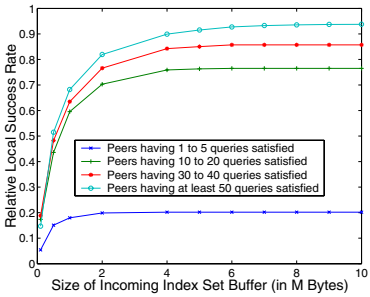
Figure 6(a) shows that CAC-SPIRP can significantly decrease the average response time of requesting peers. The response time reduction increases with the increase of the number of queries that are satisfied, because the interest localities of peers can be better exploited with an improved accuracy by gaining more experiences of content search. Figure 6(b) shows that the local relative success rate of SPIRP increases with the number of queries satisfied. For peers with more than 50 queries satisfied, the local relative success rate and the reduction of response time can be as high as about 95%.

Figure 6(a) and Figure 6(b) also show that increasing the size of incoming index set buffer helps to improve local success rate and response time. However, the local success rate and response time improve little when the buffer size is greater than 6 megabytes; and have no improvements when buffer size is greater than 10 megabytes. There are two implications for this: (1) CAC-SPIRP has a small storage requirement; (2) the locality of interests is limited and only needs a small buffer to hold.

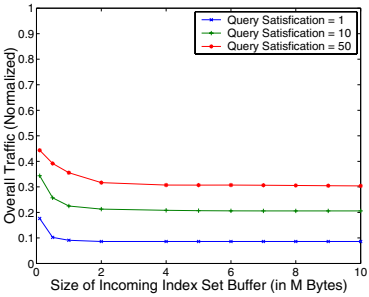
Figure 6(c) shows that the traffic reduction of CAC-SPIRP is greater than that of CAC. The reason is that CAC can reduce network traffic by limiting the



(a) The average response time per query (the query satisfaction is 50).



(b) The local relative success rate (query satisfaction is 50).



(c) The overall traffic in the P2P network.

Fig. 6. The performance of CAC-SPIRP algorithm under different query satisfactions and different sizes of incoming index set buffers. The overall traffic and average response time are both normalized by the corresponding values of flooding search.

scope of peers processing queries, and SPIRP can reduce traffic by limiting the number of outgoing queries. These two joint efforts are highly effective to reduce the overall traffic. The overall traffic reductions of CAC-SPIRP for different query satisfactions can be as high as 70% to 90%.

8 Summary

Efficient content locating in unstructured P2P networks is a challenging issue because searching algorithm designers need to consider the objectives of both improving search quality and reducing search cost, which may have conflicting interests. Existing search algorithms generally target one or the other objective. In this study, we propose CAC-SPIRP, a P2P searching algorithm aiming at both low traffic and low latency. By exploiting both the search interest localities of individual peers and the content serving locality in the peer community, our algorithm tunes the trade-off between the two objectives and achieves significant performance improvements.

Acknowledgment. We thank Beverley Yang at Stanford University, Matei Ripeanu and Adriana Iamnitchi at University of Chicago for providing their traces to us, and thank Theresa Long for proofreading this paper.

References

1. E. Adar and B.Huberman. Free Riding on Gnutella. Technical Report, Xerox PARC, August 2000.
2. F. Bustamante and Y. Qiao. Friendships that Last: Peer Lifespan and Its Role in P2P Protocols. in *Proceedings of WCW*, 2003.
3. Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham and S. Shenker. Making Gnutella-like P2P Systems Scalable. in *Proceedings of ACM SIGCOMM*, 2003.
4. J. Chu, K. Labonte, and B. N. Levine. Availability and locality measurement of peer-to-peer file systems. In *Proceedings of SPIE*, 2002.
5. Clip2 Distributed Search Solutions, http://www.clip2.com
6. Gnutella, http://www.gnutella.com
7. S. Jiang, L. Guo, and X. Zhang. LightFlood: an Efficient Flooding Scheme for File Search in Unstructured Peer-to-Peer System. in *Proceedings of ICPP*, 2003.
8. KaZaA, http://www.kazaa.com
9. LimeWire, http://www.limewire.org
10. LimeWire LLC, Ultrapeers: Another Step Towards Gnutella Scalability. http://www.limewire.com/developer/Ultrapeers.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 ICS*, 2002.
12. Morpheus, http://www.musiccity.com
13. S. Saroiu, P. K. Gummadi, and S. D. Gribble, A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of MMCN*, 2002.
14. K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems. in *Proceedings of INFOCOM*, 2003.
15. B. Yang and H. Garcia-Molina. Improving Search in Peer-to-Peer Networks. In *Proceedings of ICDCS*, 2002.