Available online at www.sciencedirect.com

SCIENCE DIRECT®

J. Parallel Distrib. Comput. 65 (2005) 729 – 742

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

Fast and low-cost search schemes by exploiting localities in P2P networks

Lei Guo[a], Song Jiang[b], Li Xiao[c], Xiaodong Zhang[a,*]

[a]*Department of Computer Science, College of William and Mary, Williamsburg, VA 23187, USA*
[b]*Performance and Architecture Laboratory, Computer and Computational Sciences Division, Los Alamos National Laboratory, Los Alamos, NM 87545, USA*
[c]*Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824, USA*

Received 19 February 2004; received in revised form 9 December 2004; accepted 26 January 2005
Available online 7 April 2005

Abstract

Existing peer-to-peer (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. Most existing work of designing and optimizing search algorithms in unstructured P2P networks addresses the trade-off between the two performance objectives. In contrast, our goal in this study is to attempt to achieve both objectives. Motivated by our observations on the content locality in the peer community and the localities of search interests of individual peers, we propose *c*ontent-*a*bundant *c*luster-*s*electively *p*refetching *i*ndices from *r*esponding *p*eers (*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*, 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*. 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 to achieve both performance objectives. Our trace-driven simulations show that CAC-SPIRP significantly improves the overall performance from both client's perspective and Internet management perspective.
© 2005 Elsevier Inc. All rights reserved.

Keywords: Peer-to-peer; Content search; Locality of content; Locality of search interest

1. Introduction

Many recent studies have shown that a tremendous amount of information is not well utilized under current client–server model in Internet systems (see e.g. [18,23]). For example, public on-line information is estimated to be 400–550 times larger than the available information accessible by traditional search engines and by hyperlinks. Peer-to-peer (P2P) systems aim to further utilize Internet information and resources, complementing the existing client–server systems. A P2P system is composed of a decentralized community of peers, where each peer acts both as a *client* who requests information and services, and as a *server* who produces and/or provides information and services.

A lot of search schemes for decentralized, unstructured P2P networks like Gnutella [9] have been proposed recently, such as [12,16,22,24,25]. The effectiveness of content search can be measured by two performance objectives that 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

* Corresponding author. Fax: +1 757 221 1717.
E-mail addresses: lguo@cs.wm.edu (Lei Guo), sjiang@lanl.gov (Song Jiang), lxiao@cse.msu.edu (Li Xiao), zhang@cs.wm.edu (Xiaodong Zhang).

0743-7315/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2005.01.007

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 iterative deepening [24] and random walking [16] target to reduce the search traffic for the system but can lead to long response time for individual peers.

The research community seems to believe that the essential issue of designing and optimizing search algorithms in unstructured P2P networks is the trade-off between the two performance objectives. However, out goal in this study aims to achieve both 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 interest topics, 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 existing trade-off between the two performance objectives, we propose *c*ontent-*a*bundant *c*luster-*s*electivity *p*refetching *i*ndices from *r*esponding *p*ers (*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*, which serves as a pool of popular objects to be frequently requested. SPIRP technique is client oriented. By SPIRP, 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 CAC is constructed, and the SPIRP technique is facilitated in each peer, retaining the merits of both CAC and SPIRP.

The contributions of our study are fourfold:

- Different from other measurement studies such as [1,19–21], we have collected query traces and index traces of a large amount of individual peers in Gnutella, and characterized the content distributions and search regularities in P2P systems.
- By exploiting the content locality in the peer community and the search interest localities of individual peers, we have proposed two efficient search techniques in unstructured P2P networks from a perspective of network management (CAC) and from a perspective of individual peers (SPIRP) respectively. In order to achieve both optimization objectives for individual peers and network management, we have combined the two techniques and proposed a fast and low cost searching algorithm, CAC-SPIRP.
- Using the traces collected from Gnutella networks and considering the population dynamics in P2P systems, we have evaluated our proposed algorithm and the two techniques, respectively. Our performance results show that unnecessary outgoing queries, network traffics, and query response times are significantly reduced, and the overheads for CAC management and index prefetching are insignificant.
- This paper provides a case study and analysis to address a fundamental issue on the design of P2P search algorithms: to improve the overall performance by optimizing both the search quality and search cost.

The remainder of the paper is organized as follows. Section 2 discusses some existing P2P searching algorithms. Section 3 presents our observations of the content serving locality in the peer community and the search interest localities of individual peers. 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

Addressing the trade-off between the search quality and search cost, researchers have proposed many search solutions in unstructured P2P systems. *Directed BFS* [24] 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 [4,16] can reduce search traffic effectively but can only give a small number of results and have long response time. *Interest-based locality* approach [22] shares the same 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. Meanwhile, the evaluation is mainly based on web traces and does not consider the population dynamics in P2P systems, thus its performance on real P2P systems is unknown. *Super-node* approach [15,25], which is adopted by Morpheus [17], KaZaA [13], and current Gnutella, limits query flooding in the domain of super peers. A super peer is a proxy and index server of a number of leaf nodes. However, the size of super peer network expands with the increase of system scale so that we still need an efficient routing algorithm for super peers.

For example, currently Gnutella has about 100,000 nodes, in which about 20,000 of them are super peers [11]. This size is even greater than the sizes of early Gnutella networks. Furthermore, a super peer is not necessarily a content-abundant peer, even the contents of all its leaf nodes are considered. Our algorithm can be applied on top of super peer architecture to further improve search performance.

3. Characterizing the localities in the peer community and individual peers

Existing measurement studies such as [1,20] investigated file distributions in P2P systems by counting numbers instead of by exploring contents. Existing algorithm studies such as [16,22] used either synthetic traces or Web traces for performance evaluations. Different from these studies, we have collected a large amount of query traces and index traces of individual peers in order to fully understand the serving regularities and access patterns of the peer community and individual peers. 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 have built two Gnutella network crawlers based on the open source code of LimeWire Gnutella [14] and conducted the following experiments to collect traces. We only consider Gnutella super peers since leaf peers in Gnutella do not accept incoming connections. The first crawler is used to collect queries in the Gnutella network. The crawler connects to a number of peers and records the query trace of each peer, including the connection establishment time, termination time, and the time and search criterion of each query it sends. We ran 10 crawlers for 4 days and recorded 409,129 queries of 25,764 different peers altogether. Both the number of queries a peer sends and the duration of a peer's connection session follow heavy tail distributions. We randomly selected 1600 peers and their corresponding queries (total 25,093 queries) in our traces as the *P2P client set* for our study.

The second crawler is used to collect the indices of shared files of Gnutella peers. The crawler randomly connects to a number of peers, and sends a "ping" message to each neighbor to get the number of files it is sharing. Then the crawler sends an indexing query [10] to get the index of shared files. The crawler closes the connection and connects to another peer after receiving the entire index from the peer.

We ran 40 index crawlers for 4 days and collected the entire indices of 18,255 different peers, in which each peer is identified by its (Globally Unique Identifier (GUID)—which is attached in the query response messages) instead of its IP address, since many peers use Dynamic Host Configuration Protocol (DHCP) or Network Address Translation (NAT) to access Internet. At the same time, we estimate that there are 37% free-riders in Gnutella networks based on the "pong" messages we collected. 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

Study [20] shows a small percentage of peers share much more number of files than other peers in P2P systems. Paper [5] studied 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, from the perspective of content serving, a peer's ability to contribute contents depends on both the contents it is sharing and the query distributions in P2P systems. 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. Fig. 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 1000 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. Fig. 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 2500 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 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*.

Fig. 1(c) shows 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. 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.

We have studied the diversity of peers' content serving capacity in P2P systems. Figs. 2(a) and (b) show the

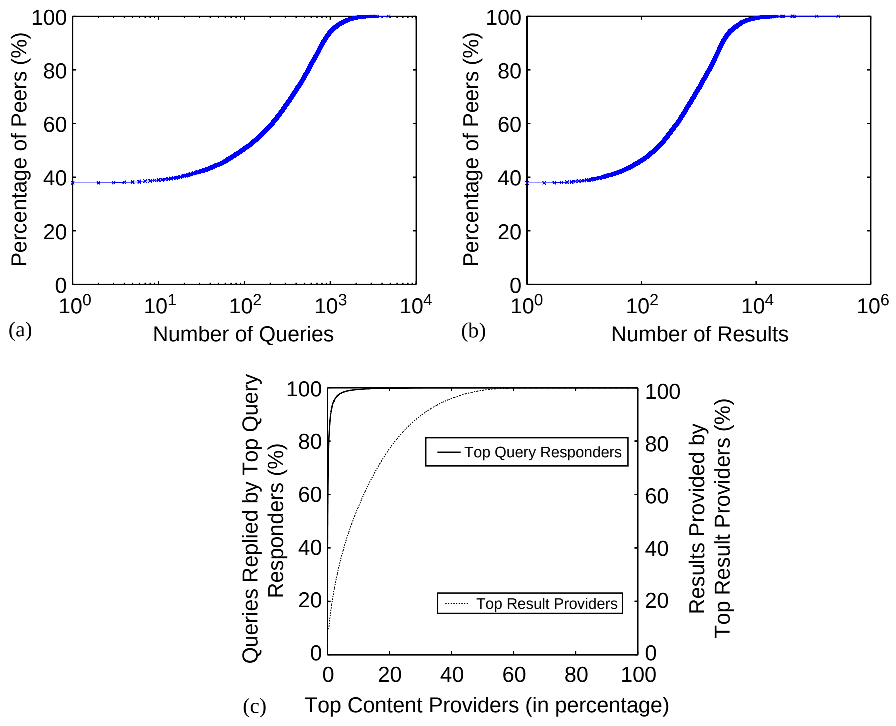

Fig. 1. The skewness of Gnutella peers' abilities to reply queries and to provide results: (a) the CDF of the number of queries that peers can reply, (b) the CDF of the number of results that peers can provide and (c) the cumulative contributions of top query responders and result providers.

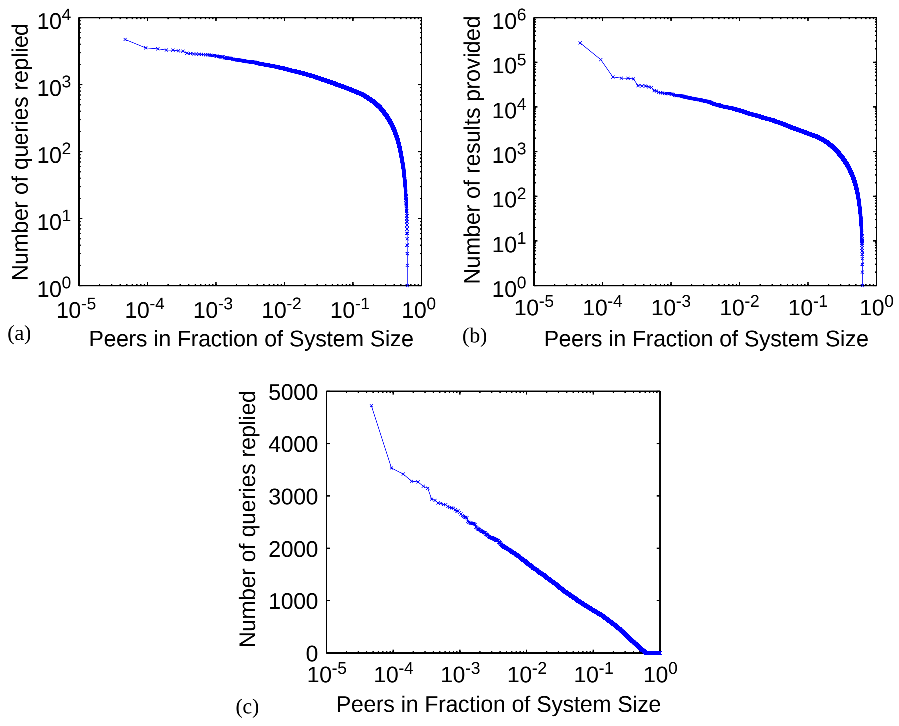

Fig. 2. The distribution of peers' capacity in query serving: (a) the query replying (in log–log scale), (b) the results providing (in log–log scale) and (c) the query replying (only x is in log scale).

distribution of peers' capacity to reply queries and to provide results in a log–log scale, respectively. We find that it is easy to distinguish those content-abundant peers from content-scare peers: there is a sharp decrease in both plots, indicating the gap between these two kinds of peers. Both distributions do not follow Zipf-like distribution such as the reference locality in Web systems [2]. Had the data distribution been Zipfian, both plots should be linear in a log–log scale. Fig. 2(c) re-plots the distribution of peers' capacity to reply queries in a log scale on *x*-axis only. We find this plot can be well fitted in a straight line. The distribution of peers' capacity to provide results is not linear in an *x*-axis log scale, but much more linear than a Zipf curve. Our discovery means that the diversity of peers' capacity in P2P systems is much smaller than that of Internet Web servers, which has two implications: (1) the diversity of content requesting and serving is much lower in P2P world than in the Web world; (2) the transient behavior of peers cannot affect the performance of P2P systems as long as the system population is big enough.

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. We also find the diversity of content serving capacity of those content-abundant peers is small.

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 into 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 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.

Fig. 3 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 Fig. 3(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. Fig. 3(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. Figs. 3(a) and (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.

3.4. Summary of our observations and motivations

We summarize our observations and motivations as follows:

- Strong locality of content exists in the peer community: a small percentage of peers, which are called top content providers in the community, account for most content contributions in the system.
- The diversity of content serving capacity is small for those content-abundant peers in P2P systems. Thus, the collection of top content providers can provide highly qualified service.
- The search patterns of individual peers show strong localities of search interests. For example, a requesting peer shares the same interests with its top query responders.
- The content locality in the peer community implies that the majority of peers are not able to provide satisfactory query results. Thus, randomly propagating queries in P2P networks are very likely to reach those peers who have limited ability or inability to serve contents, significantly wasting network bandwidth. This motivates us to design an algorithm that objectively sends queries to those top content providers in the peer community.
- The interest localities of individual peers indicate that the requesting peers frequently access their top content providers for the contents of their interests from time to time. This motivates us to let these requesting peers exploit their interest localities by prefetching the content

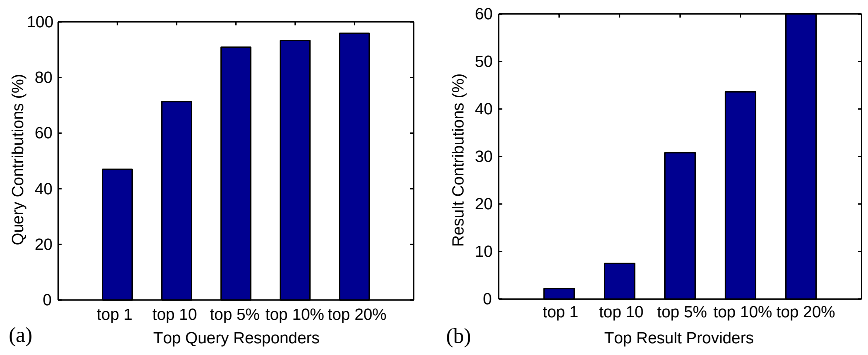

Fig. 3. The average contributions (in percentage) of top query responders and top result providers for peers having at least 10 replied queries: (a) The average query contributions (in percentage) of top 1, top 10, top 5%, top 10%, and top 20% query responders and (b) The average result contributions (in percentage) of top 1, top 10, top 5%, top 10%, and top 20% result providers.

indices of their top content providers, so that content search can be performed locally, significantly reducing the response latency.

4. CAC technique: constructing CAC

The basic idea of CAC technique is to have a collection of content-abundant peers in the peer community be self-organized into a CAC to actively serve contents for the entire P2P network. 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 cannot 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.

4.1. CAC initialization

The initialization of the CAC can be implemented by using a *bootstrap site*, a permanent host in the P2P system like the host-cache sites in Gnutella. Initially, all cluster peer candidates connect to the bootstrap site and report their locations. Then the bootstrap site sends a list of peer locations to each of them. Then these peers self-organize into a cluster (the CAC) by randomly selecting about 4–8 peers from the list as their neighbors.

4.2. 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 networks. Our simulations show that CAC has no strict requirements on these two parameters (see Section 7.3.1).

CAC technique is modular and can be easily deployed on top of any P2P overlays. The CAC 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 link, the other is the CAC overlay link. Each peer in the system is assigned a *level*: the level of each CAC peer is defined as 0, and the level of a 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.

4.3. Query routing

A query is routed from higher-level peers to lower-level peers until reaching the CAC, shown in Fig. 4(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 Fig. 4(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

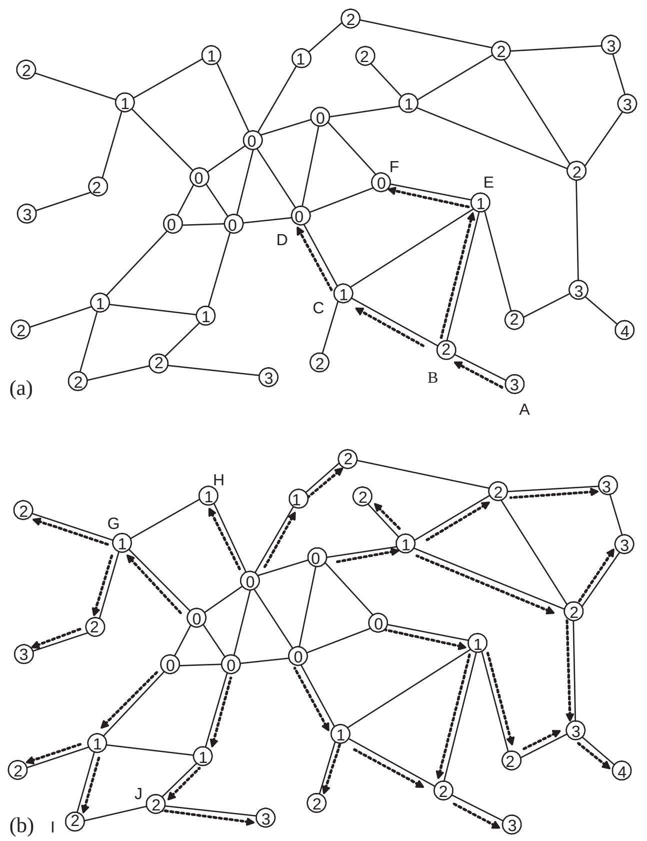

Fig. 4. The up-flowing and down-flooding operations. Each circle denotes a peer in the P2P network, and the number in the circle is the level of this peer. The bold lines denote the CAC links and the thin lines denote the original P2P links. (a) Up-flowing operation: shows the routing paths of a query 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) Down-flooding operation: 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.

propagating queries, reducing a great amount of unnecessary traffic.

CAC has several advantages over the super peer structure due to several major differences between the two structures. Firstly, CAC is content-based, which has much richer resources than index-based super peers. Secondly, in the super peer structure, the index services are provided via individual connections between peers and one or a few super peers. If the connected super peer(s) fail, the peers will be out of services. In contrast, CAC is significantly more robust in query routing. The normal peers connect to CAC, where the failures of individual CAC peers would not affect to the normal peers, because there will always be available CAC peers to provide services. Finally, the routing paths for up-flowing and down-flooding operations are on top of the P2P overlay and self-adaptive. As long as the P2P overlay is connected, the routing path will always exist.

4.4. System maintenance

The CAC is maintained in a proper size that can be predefined or dynamically refreshed based on the query success rate in CAC. In the static case, 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. In the dynamic case, each CAC peer maintains the success rate of queries that are up-flowed to it, and uses the average success rate of its neighbors within one-hop in the CAC overlay to estimate the required CAC size. The maintenance overhead is trivial since the constraint on CAC size is not hard to get good performance, as shown in our evaluation (see Section 7.3.1).

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 have the priority to receive and respond queries. In CAC technique, each CAC peer randomly selects 5% queries from all queries up-flowed to it, and down-floods them in the P2P overlay in addition to flooding them in the CAC simultaneously. Both CAC and non-CAC peers evaluate their service qualities based on these queries only. In P2P systems such as Gnutella, each peer receives more than 1000 queries per minute, and our experiment shows that 100–200 queries are sufficient to identify a content-abundant peer.

CAC peers who cannot provide qualified services for some period of time and overloaded peers leave the CAC to become normal peers. 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.

The performance of CAC is stable in spite of the population dynamics due to the transient coming and going of CAC peers. As shown in Section 3.2, the diversity of content serving capacity of peers in P2P networks is small, meaning that there is no peer significantly more powerful than other peers in the CAC. Thus, CAC performance cannot be affected as long as the CAC size is not very small.

4.5. The incentive for becoming CAC peers

CAC peers support more search workloads for the system than common peers. Although the search workload is relatively lighter than downloading workload, an incentive mechanism would be beneficial to encourage CAC participation and improve the robustness of CAC system. Effective incentive mechanisms that encourage peers to contribute bandwidth for P2P downloading service have been proposed, such as BitTorrent [7]. Due to the priority to forward queries and responses, CAC peers have more knowledge on the content distributed in the system. These information can be used to build content index to speed up query processing, which is the basic idea of SPIRP technique (see Section 5) and forms an incentive to become CAC peers.

5. SPIRP technique: 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. 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 and the value is the responder's meta data. The format of a responder's meta data is 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.

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.

5.2.1. 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.

5.2.2. 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.

5.2.3. 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.

5.2.4. 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. The CAC-SPIRP algorithm: 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

Table 1
The data structure of responder's meta data

Field	IP addr.	Port	Is cached	Priority	Index size	Timestamp	Expire time	Update time	Other fields
Bytes	4	2	2	4	4	4	4	4	16

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 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. Since outgoing queries are always routed to CAC first, the bandwidth consumption of CAC-SPIRP is at least as low as that of CAC. Each peer also has the ability to selectively prefetch indices for locality of interests, decreasing search latency and further reducing bandwidth consumption.

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 [20] shows that the session duration of peers follows heavy tail distribution, where the duration median is about 60 min. This study is consistent with our observations about the connection durations between the query collection crawler and Gnutella peers in Section 3.1. Study [3] further shows the lifespan of peers follows the Pareto distribution. Different from the simulations of existing studies such as [22], 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 [20]. Assuming the total number of on-line peers in a P2P system is constant, we replace a peer with a randomly chosen new peer in the P2P server set when its session terminates and the off-line peer is considered as a new peer for future use. We use the topology traces provided by Clip2 Distributed Search Solutions [6] and University of Chicago in our simulations. Due to page limit, we only present the simulation on a Gnutella snapshot of 6946 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 specifies (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 the incoming index set locally is defined as 0. 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 two techniques separately to show their corresponding effectiveness, and then evaluate our proposed algorithm to show the performance of their combination. We chose the 1600 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.

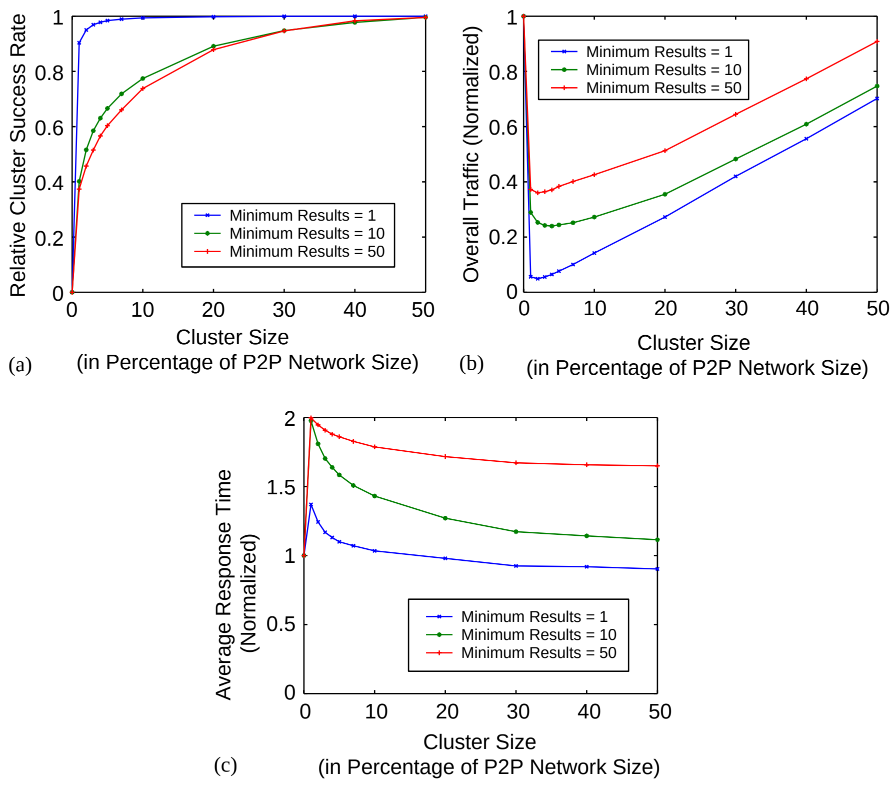

Fig. 5. The performance of CAC technique under different query satisfactions and different sizes of clusters in which the cluster peers are those top query responders. The overall traffic and average response time are both normalized by the corresponding values of flooding search: (a) the cluster relative success rate, (b) the overall traffic in the P2P network and (c) the average response time per query.

The overall traffic and average response time presented in the rest of the paper are normalized by the corresponding performance values of flooding search under the same trace-driven simulation. For example, if the overall traffic is reported as 0.5, the real traffic is 50% of that of the flooding search.

7.3.1. *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.

Fig. 5(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. Fig. 5(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. Compared with super peer approach, assuming that the population of super peers in P2P networks is about 20% of all peers in the system (see Section 2), we can estimate that CAC approach has only 28%, 69%, and 75% traffic in super peer search solution under different user satisfaction requirement of 1, 10, and 50 results, respectively.

The response time of CAC technique is not so good. Fig. 5(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

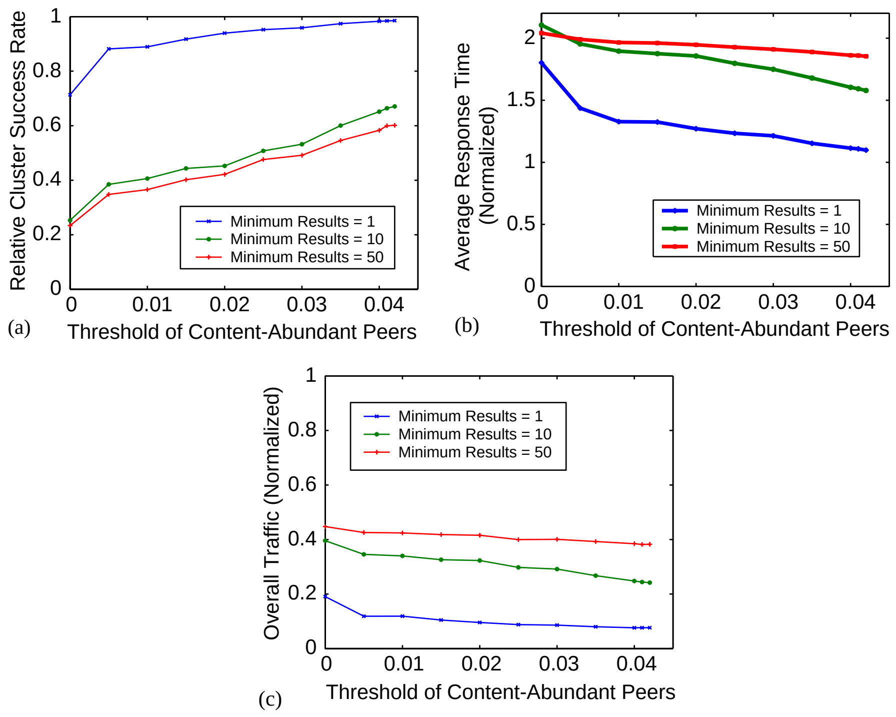

Fig. 6. The performance of CAC technique under different query satisfactions and different quality thresholds of content-abundant peers. The cluster size is set to 5% of the P2P network size. The overall traffic and average response time are both normalized by the corresponding values of flooding search: (a) the cluster relative success rate, (b) the overall traffic in the P2P network and (c) the average response time per query.

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.

Fig. 6 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 cannot affect the system performance seriously.

We have also compared the performance of CAC with iterative deepening [24] and expanding ring [16] 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 those of both algorithms under the same conditions. We do not present the figures due to page limit.

7.3.2. *Performance evaluation of SPIRP technique*

In this experiment, we maintained the incoming index set of each requesting peer in a fixed size of buffer and measured the overall network traffic, average response time, and the local relative success rate in the incoming index set under different query satisfactions and different sizes of buffers. The results are presented in Fig. 7.

Fig. 7(a) shows that 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. Fig. 7(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 more than 95%.

Figs. 7(a) and (b) also show that increasing the size of incoming index set buffer helps to improve local success

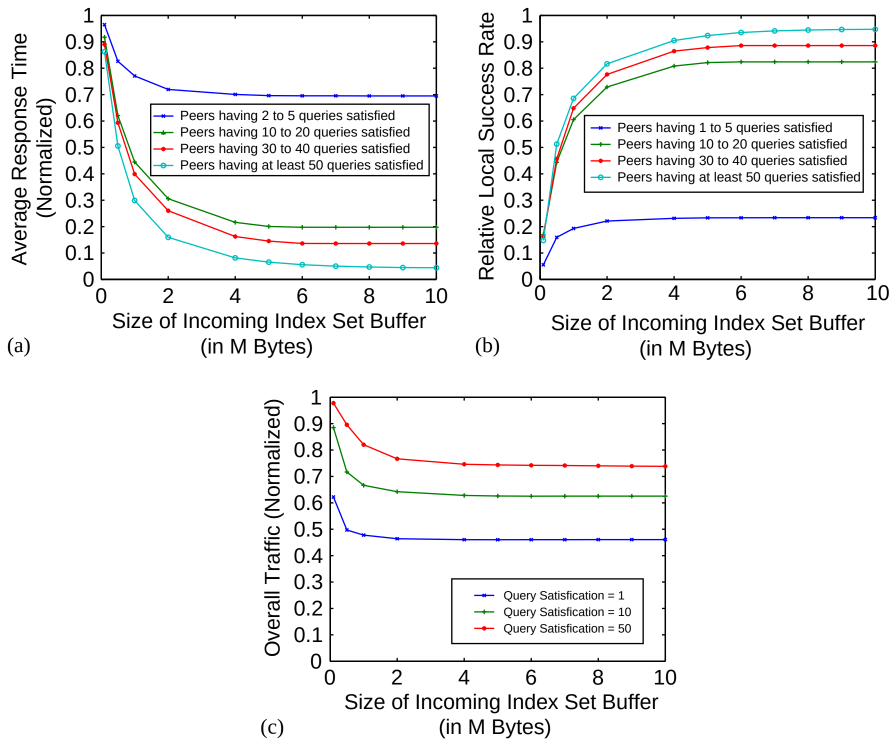

Fig. 7. The performance of SPIRP technique under different query satisfactions and different sizes of incoming index set buffers. The overall traffic and average response time are both normalized by the corresponding values of flooding search: (a) the average response time per query (the query satisfaction is 50), (b) the local relative success rate (the query satisfaction is 50) and (c) the overall traffic in the P2P network.

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) SPIRP has a small storage requirement; (2) the locality of interests is limited and only needs a small buffer to hold.

SPIRP not only can improve user's search performance, but also an decrease the overall traffic in P2P networks. Fig. 7(c) shows the traffic reduction can be as high as more than 50% for query satisfaction of 1 result. This is because the index prefetching traffic is small compared with the flooding traffic of outgoing queries that are avoided. Our studies show that the index prefetching cost is only about 3.8–9.4% of the overall traffic in flooding algorithm for different query satisfactions from 1 to 50. Meanwhile, the greater the query satisfaction is, the smaller the traffic reduction will be. This confirms the existence of the conflicts between the search quality and the search cost.

7.3.3. Performance evaluation of CAC-SPIRP

CAC significantly reduces the overall network traffic at the expense of performance degradation in response time. SPIRP reduces the response time remarkably but the decrease of the network traffic is not satisfactory. 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 both objective. 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 Fig. 8.

Fig. 8(a) shows that CAC-SPIRP can reduce the response time up to 90% when the number of satisfied queries reaches to a certain number (e.g., 40–50), as effectively as SPIRP. When the number of satisfied queries is too small, the response time is worse than SPIRP and the flooding algorithm but still better than CAC technique.

Comparing Fig. 8(b) with Fig. 7(b), we can see that the local relative success rate in the incoming index set buffer of CAC-SPIRP is slightly lower than that of SPIRP. The reason is that peers with the same interests as an individual peer might not be in the CAC cluster since they are not necessarily content-abundant peers. However, this difference is very small (less than 3%).

Fig. 8(c) shows that the traffic reduction of CAC-SPIRP is greater than those of both CAC and SPIRP. The reason is that CAC can reduce network traffic by limiting the 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

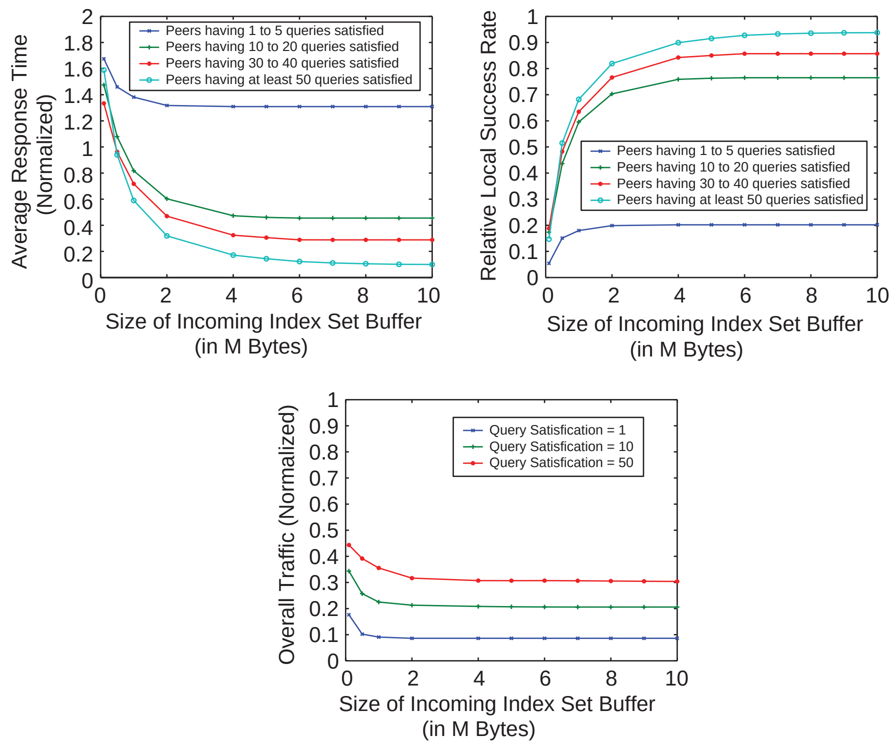

Fig. 8. The performance of CAC-SPIRP algorithm under different query satisfactions and different sizes of incoming index set buffers. The overall traffic and average response time are both normalized by the corresponding values of flooding search: (a) the average response time per query (the query satisfaction is 50), (b) the local relative success rate (query satisfaction is 50) and (c) the overall traffic in the P2P network.

overall traffic reductions of CAC-SPIRP for different query satisfactions can be as high as 70–90%.

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 locality in the peer community, we aim at achieving both objectives for significant performance improvements. Performance evaluation shows the effectiveness of our algorithm and system organization.

9. Acknowledgments

This work is supported in part by the US National Science Foundation under grants CNS-0098055, CCF-0129883 and CNS-0405909. Some preliminary results of this work have been presented in [8]. We thank Beverley Yang of Stanford University, Matei Ripeanu and Adriana Iamnitchi at University of Chicago for providing their traces to us, and thank Theresa Long for reading this paper and her comments.

References

[1] E. Adar, B. Huberman, Free riding on Gnutella, Technical Report, Xerox PARC, August 2000.

[2] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, Web caching and Zipf distributions: evidence and implications, in: Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '99), New York, USA, March, 1999.

[3] F. Bustamante, Y. Qiao, Friendships that last: peer lifespan and its role in P2P protocols, in: Proceedings of the Eighth International Workshop on Web Content Caching and Distribution, Hawthorne, NY USA, 2003.

[4] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, S. Shenker, Making Gnutella-like P2P Systems Scalable, in: Proceedings of ACM SIGCOMM 2003, Karlsruhe, Germany, 2003.

[5] J. Chu, K. Labonte, B.N. Levine, Availability and locality measurement of peer-to-peer file systems, in: Proc. ITCom: Scalability and Traffic Control in IP Networks II Conferences, 2002.

[6] Clip2 Distributed Search Solutions, http://www.clip2.com

[7] B. Cohen, Incentives build robustness in BitTorrent, in: Proceedings of the First Workshop on the Economics of Peer-to-Peer Systems, Berkeley, CA, USA, June, 2003.

[8] L. Guo, S. Jiang, L. Xiao, X. Zhang, Exploiting content localities for efficient search in P2P systems, in: Proceedings of the 18th International Symposium on Distributed Computing, (DISC 2004), Amsterdam, Netherlands, October, 2004.

[9] Gnutella, http://www.gnutella.com
[10] Gnutella 0.6, http://rfc-gnutella.sourceforge.net
[11] http://www.limewire.com/english/content/netsize.shtml
[12] S. Jiang, L. Guo, X. Zhang, LightFlood: an efficient flooding scheme for file search in unstructured peer-to-peer system, in: Proceedings of the 2003 International Conference on Parallel Processing (ICPP 2003), Kaohsiung, Taiwan, China, 2003.
[13] KaZaA, http://www.kazaa.com
[14] LimeWire, http://www.limewire.org
[15] LimeWire LLC, Ultrapeers: another step towards Gnutella scalability, http://www.limewire.com/developer/Ultrapeers.html
[16] Q. Lv, P. Cao, E. Cohen, K. Li, S. Shenker, Search and replication in unstructured peer-to-peer networks, in: Proceedings of the 16th ACM International Conference on Supercomputing (ICS'02), New York, USA, 2002.
[17] Morpheus, http://www.musiccity.com
[18] S. Raghavan, H. Garcia-Molina, Crawling the hidden web, in: Proceedings of the 27th VLDB Conference, 2001.
[19] S. Saroiu, K. Gummadi, R. Dunn, S. Gribble, H. Levy, An Analysis of Internet Content Delivery Systems, OSDI, 2002.
[20] S. Saroiu, P.K. Gummadi, S.D. Gribble, A measurement study of peer-to-peer file sharing systems, in: Proceedings of MMCN 2002, San Jose, USA, 2002.
[21] S. Sen, J. Wang, Analyzing peer-to-peer traffic across large networks, in: Proceedings of ACM SIGCOMM Internet Measurement Workshop 2002.
[22] K. Sripanidkulchai, B. Maggs, H. Zhang, Efficient content location using interest-based locality in peer-to-peer systems, INFOCOMM, 2003.
[23] The Deep Web:Surfacing Hidden Value http://www.completeplanet.com/Tutorials/DeepWeb/
[24] B. Yang, H. Garcia-Molina, Improving search in Peer-to-peer networks, in: Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS 2002), Viennam Austria, 2002.
[25] B. Yang, H. Garcia-Molina, Designing a super-peer network, in: Proceedings of the 19th International Conference on Data Engineering (ICDE 2003), Bangalore, India, 2003.

Lei Guo received his BS degree in space physics and MS degree in computer science from the University of Science and Technology of China in 1996 and 2002, respectively. He is a PhD candidate in computer science at the College of William and Mary. His research interests are in the areas of distributed systems, peer-to-peer systems, multimedia systems, and Internet measurement.

Song Jiang received the BS and MS degrees in computer science from the University of Science and Technology of China in 1993 and 1996, respectively, and received his PhD degree in computer science from the College of William and Mary in 2004. He is a Postdoctoral Research Associate at the Los Alamos National Laboratory, developing next generation operating systems for high-end systems. He received the S. Park Graduate Research Award at the College of William and Mary in 2003. His research interests are in the areas of operating systems, computer architecture, and distributed systems.

Li Xiao received the BS and MS degrees in computer science from Northwestern Polytechnic University, China, and the PhD degree in computer science from the College of William and Mary in 2002. She is an assistant professor of computer science and engineering at Michigan State University. Her research interests are in the areas of distributed and Internet systems, system resource management, and design and implementation of experimental algorithms. She is a member of the ACM, the IEEE, the IEEE Computer Society, and IEEE Women in Engineering.

Xiaodong Zhang received his BS degree in electrical engineering from Beijing Polytechnic University in 1982, MS and PhD degrees in computer science from University of Colorado at Boulder in 1985 and 1989, respectively. He is the Lettie Pate Evans Professor of Computer Science and the Department Chair at the College of William and Mary. He was the Program Director of Advanced Computational Research at the US National Science Foundation from 2001 to 2003. He is a past editorial board member of *IEEE Transactions on Parallel and Distributed Systems*, and currently serves as an associate editor of *IEEE Transactions on Computers and IEEE Micro*. His research interests are in the areas of parallel and distributed computing and systems, and computer architecture.