A Two-Level Semantic Caching Scheme for Super-Peer Networks

Paweł Garbacki[1], Dick H. J. Epema[1], and Maarten van Steen[2]

[1] Faculty of Electrical Engineering,
Mathematics, and Computer Science
Delft University of Technology,
The Netherlands
{p.garbacki,d.h.j.epema}@ewi.tudelft.nl
[2] Department of Computer Science
Vrije Universiteit Amsterdam,
The Netherlands
steen@cs.vu.nl

Abstract. Some recent measurement studies of file-sharing peer-to-peer networks have demonstrated the presence of semantic proximity between peers and between shared files. This observation may be used for improving the performance of searching by introducing semantic caches. One type of such caches links peers that are interested in similar files. The query routing mechanism uses this information by forwarding queries first to peers which are semantically close. The second type of semantic caches groups similar content instead of similar nodes. In this paper we show how to combine both methods by introducing a two-level caching infrastructure based on super-peers. The super-peers in our system cache pointers to files recently requested by their client peers. The client peers, on the other hand, constantly look for the super-peers that are most suitable for them. We propose a simple, yet powerful cache management policy that guarantees high cache hit ratios also for the less popular files. Further, we discuss the design choices and optimizations of the presented model. Finally, we evaluate our system versus the symmetric network that uses only one level of semantic caches.

1 Introduction

Peer-to-peer content sharing has become very popular in the last few years, and is nowadays the biggest consumer of Internet bandwidth [19, 23]. The success of P2P technology has attracted the interest of the research community, which has resulted in many improvements to the existing protocols as well as completely new P2P designs [6, 14, 16]. However, the dominating P2P content sharing networks seem to completely ignore the dynamic properties of the shared data such as file popularities or flash crowds. Search is either blind [1, 3], independent of the query, or based on static aspects of the content [21, 22, 25, 29] such as file hashes. In this paper we propose an optimization that can be integrated with both these searching schemes improving their performance by taking advantage of the semantic correlation [8, 10] between peers as well as shared files. The problem that needs to be solved here is the efficient management of semantic information. We tackle this issue by introducing a hierarchy into the network in the form of super-peers.

Super-peer networks [28] occupy the middle-ground between centralized and entirely symmetric P2P networks. Super-peers are selected nodes with extra capabilities, but also extra duties in the network. A super-peer acts as a server to a dynamic subset of weak (ordinary, client) peers. Weak peers submit queries to their super-peers and receive results from them. Super-peers are connected to each other forming an overlay network of their own, submitting and answering requests on behalf of the weak peers.

Creating connections between weak peers and super-peers can be seen as peer clustering; a cluster in this context is simply the set of peers that are connected to the same super-peer. The semantic structure in the shared content and the patterns observed in successive searches made by individual nodes can be used as the clustering criterion. Grouping together peers with similar interests under one super-peer increases query locality, and as a consequence improves the performance of search [17].

A semantic structure can be defined in several ways. Because content classification is generally a difficult problem [12], the most common approach is to identify semantic groups explicitly [7, 15, 17, 18]. In this paper, we take an alternative approach, trying to detect semantic relationships between peers automatically based on user interests [24]. The potential of methods in this category was shown in [27].

Clustering may be applied either to peers or to content. In the first case, peers with similar interests are grouped under one super-peer. Content clustering, in turn, results in pointers to files with similar request patterns cached at the same super-peer. In our approach we combine both methods, and the proposed architecture assumes the existence of caches of two types. Weak peers keep lists of super-peers that proved in the past to be in some way most suitable for them. Super-peers index files that were recently requested by their weak peers. The mutually re-enforcing dependency between the cache management policies at the weak peers and the super-peers results in a convergence to a Pareto optimal [11] steady state in which cache modifications occur only occasionally.

The rest of the paper is organized as follows. Section 2 describes in detail the system architecture, the performance of which is assessed in Section 3. The paper concludes in Section 4 by exploring some opportunities for future work.

2 Detailed system model

In this section we describe in detail the concept of the super-peer network based on two-level semantic caches.

2.1 Design considerations

We take several design decisions that make our architecture different from other approaches addressing the problem of managing semantic information in P2P systems. For each decision we provide a brief rationale for adopting it.

Caching is the most commonly used technique for discovering and exploiting semantic dependencies between peers and shared content. Most of the existing P2P infrastructures based on caching of the search results assume the existence of homogeneous caches at all nodes. Such approaches seem to ignore the fact that caching can be efficient only if a sufficient number of queries is generated. Each cache has a warm-up phase in which it collects information about the access characteristics of the stored items. A new peer joining the system has to first build its own cache before it can profit from the semantic structure of the network. We solve this problem by placing shared caches at a set of selected (super-)nodes. These caches are used (shared) by many peers at the same time. A peer that joins the system is automatically bound to one or more super-peers and can immediately use the information collected by these super-peers.

The question that arises then is how replacing a number of local caches with one shared cache will influence the hit ratios? According to [5], the average hit ratio of a cache capable of storing n items is proportional to $log(n)$. Let's assume that we replace caches of size n located at m peers with a single shared cache of size n. The worst case scenario occurs when peer interests are distinct (peers search completely different files). Assuming that each peer contacts the shared cache with the same frequency, the cache hit ratio is proportional to $log(n/m)$, which can be written as $log(n) - log(m)$. The formula comes from the observation that each peer uses a portion of size n/m of the shared cache. We conclude that the cache hit ratio decreases logarithmically with m. It should be added that in a real system the size of the shared cache is higher than n. Furthermore, many peers share interests for the same files which results in much higher hit ratios.

Introducing super-peers into the architecture of a P2P system usually implies a serious limitation of the weak peer autonomies. We try to decrease the extent of this problem by establishing loose relationships between the weak peers and the super-peers. The weak peers are free to decide which super-peers they are connected to. This decision is made locally and independently from other peers. The super-peer selection rule has however a global property of grouping peers with similar interests under one super-peer.

2.2 System architecture

In this section we describe the architecture of the super-peer network incorporating a two-level caching scheme.

Figure 1 presents the data structures used in our system. The information stored at a node depends on the type of this node. Each weak peer p has a *super-peer cache* $p.S$, which contains the identities of super-peers (e.g., their IP addresses and port numbers). Each super-peer s has a *file cache* $s.F$ of pointers to files stored at some peers.

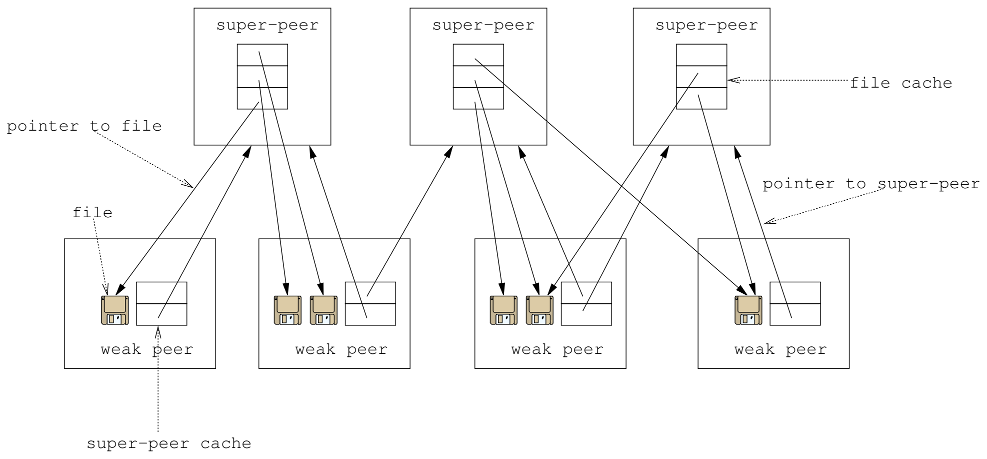

Fig. 1. The two-level caching scheme.

All items in the super-peer and file caches are assigned *priorities*, which are non-negative integer numbers. The priority determines the importance of a particular item, the higher the better. The initial priority assigned to a data item when it is added to the cache and the way the priority is modified upon a cache hit are determined by the caching policy (see Section 2.5). There are two situations when the priorities are taken into account. First, when the cache capacity is exceeded, the item with the lowest priority is removed. Second, the priorities are used for optimizing query routing. Details are presented in Section 2.3.

The super-peers are interconnected with one of the standard P2P networks, which we leave unspecified. We require however that the probability that the search succeeds is high if the requested information is possessed by only one of the super-peers. Examples of protocols satisfying this criterion are Gnutella and epidemic-based approaches such as SCAMP [9].

Whenever a weak peer initiates a search, it first checks the file caches of the super-peers known to it. If the file is not found in one of these caches, a system-wide search in the super-peer network is initiated. The pointer to the located file is then cached by one of the super-peers known to the weak peer that initiated the search. Note that in this scheme only the files that are known to at least one of the super-peers can be located. To increase the probability of finding the file we could perform a search in the entire P2P network instead of among the super-peers only. As we show in Section 3, such an expanded search is usually not needed because most of the files are indexed by at least one super-peer.

The performance of the search is determined by the cache hit ratio. Peer p should have in its cache super-peers indexing content which is closest to p's interests. This is achieved by preferring super-peers that provided in the past the highest number of positive responses to the queries submitted by p.

By allowing super-peers to cache pointers to the files recently requested by their weak peers, we exploit a simple, yet powerful principle called *interest-based locality* [24]. Interest-based locality postulates that if two peers are interested in the same file, it is very likely that more of their requests will overlap.

2.3 The search protocol

The following pseudocode describes the actions taken by peer p searching for file f.

```
peer_search(p : peer, f : file_name)
1:  for (s in p.S ordered according to decreasing priorities)
2:    q := super-peer_local_search(s,f)
3:    if (super-peer_local_search succeeded) // f was found
4:      t := s
5:      break
6:  if (f was not yet found)
7:    s := super-peer in p.S selected randomly with probability
           proportional to its priority in p.S
8:    <q,t> := super-peer_search(s,f)
9:    if (super-peer_search did not succeed) // f was not found
10:     return ERROR "File f not found"
11: if (p.S contains t) then
12:   increase the priority of t in p.S
13: else
14:   insert t into p.S
15: return q // peer that has file f
```

In the above algorithm, peer p first tries to locate file f in the local caches of the super-peers in its super-peer cache. Note that p starts from the super-nodes with the highest priorities. If the file was not found in this way (line 6), the search request is forwarded to one of the super-peers in $p.S$ selected randomly based on its priority (line 7). This super-peer takes care of locating file f and returns a pair $< q, t >$, where q is a peer that shares f and t is a super-peer that has a pointer $< f, q >$ in its cache. Finally, peer p updates the priority of t in its cache (lines 11—14).

The algorithm of the super-peer local search is straightforward.

```
super-peer_local_search(s : super-peer, f : file_name)
1:  if (an entry <f,q> exists in cache s.F)
2:    increase the priority of <f,q> in s.F
3:    return q
4:  else
5:    return ERROR "File f not found"
```

The local search succeeds only if a pointer to file f is in the local cache of s (line 1). Before returning the location of the file, the super-peer increases the priority of the corresponding cache item (line 2).

The global super-peer search protocol is as follows.

```
super-peer_search(s : super-peer, f : file_name)
1:   perform a search in the super-peer network trying to locate
       a super-peer t which has an entry <f,q> in its cache
2:   if (the search did not succeed)
3:     return ERROR "File f not found"
4:   else
5:     insert <f,q> into s.F
6:   return <q,t>
```

The function `super-peer_search` is a wrapper for the search in the underlying super-peer network (line 1). Upon receipt of the search results, the pointer to the located file with the identity of the content provider peer q are added to the cache $s.F$ (line 5) and the pair $< q, t >$ is returned (line 6).

2.4 Insert protocol

Our insert protocol is very simple. Each peer p once in a while sends information on the files which it possesses to one of the super-peers in its super-peer cache. This super-peer is selected randomly with a probability proportional to its priority in p's super-peer cache.

2.5 Cache management

The super-peer and file caches are controlled differently. Whenever peer p receives positive feedback from super-peer s, the priority of s in cache $p.S$ is incremented by one which leads to the *in-cache least frequently used* (LFU) [5] cache management policy. The advantage of this method is the inherent memory property, which means that the priority of the super-peer is based on the amount of successful feedback it has provided in the past. The priority changes slowly, so one positive response from a completely unknown super-peer will not influence much the peer to super-peer assignment (this would be the case if we used one of the memoryless strategies [13] such as *least recently used* — LRU).

The management policy of the file caches should have two properties. First, it should enable the super-peers to adapt fast to the changing needs of their client peers. This is important, particularly in the initial stage of the super-peer lifetime, when it is contacted by random peers. Second, the cache management strategy should not ignore unpopular files. The cost of the search in the underlying super-peer network incurred by a cache miss is much higher for an unpopular file. In flooding protocols like Gnutella, looking for a file with only a few replicas in the whole network is much more expensive than searching for a popular file. The strategy proposed here is very simple. If the recently accessed file is in the cache, we increment its priority by one. Otherwise, we add this file with priority one higher that the highest priority of all cached items. This approach combines the fast adaptation of LRU with the memory property of LFU (in the remainder of the paper we call this caching method the *mixed* strategy). The high initial priority of the inserted item and the slow adaptation of the priorities of items in the cache prolong the caching period of less popular items.

3 Performance evaluation

This section presents the results of the evaluation of the system described in Section 2. We start with a description of the method which we used for modeling of the semantic structure. Then we explain the simulation parameters. Finally, we compare the performance of our system with the one offered by a system that uses only one level of semantic caches.

3.1 Model of the semantic structure

In our experiments we use a synthetic data model similar to the one introduced in [27]. This model assumes the existence of a number of (semantic) types for both files and peers labeled by $n \in \{1, \ldots, N\}$, with N denoting the number of types. The number of files of type n is denoted by d_n, and the number of peers associated with type n is denoted by u_n.

Each peer periodically generates a request. The target file satisfying this request is selected randomly, according to a distribution that depends on the peer's type only. This distribution is specified by two parameters: the probability $p_n(m)$ that a request generated by a peer of type n will be targeted at a file of type m, and the probability $q_m(k)$ that a request for a file of type m will target the k-th file of this type. The distribution of $q_m(k)$ follows Zipf's distribution, which has been found to occur in real data traces [24].

The formulas for $p_n(m)$ can be written as follows

$$\begin{cases} p_n(m) = \frac{1-\alpha}{m}/Z & , m = 1, \ldots N,\ m \neq n, \\ p_n(n) \ = (\alpha + \frac{1-\alpha}{n})/Z\ , \end{cases} \tag{1}$$

where Z is the normalizing constant chosen so that $\sum_m p_n(m)$ equals 1. Z is given by $(1-\alpha)H_N + \alpha$, where $H_N = \sum_{m=1}^{N} 1/m$ is the N-th harmonic number. Note that Z does not depend on n. The parameter α characterizes how strong the interest of users is for files of their own type.

Furthermore, we assume that our model satisfies two conditions. First, the number u_n of users of type n follow Zipf's law. Second, we require that the numbers of files d_n are all equal to M, independent of n. This second assumption may be controversial, but assuming that d_n follows Zipf's law as in [27] does not seem to be appropriate. This claim is based on our experience gained during our long-term measurements of BitTorrent, which is nowadays (March 2005) the most popular P2P network [20]. According to the statistics provided by one of the biggest `.torrent` distribution sites [4] in March 2005, the most popular content categories, which are movies and games, have much fewer items than the relatively unpopular music files. In this case the uniform distribution of d_n

seems to be a better approximation of the real situation than Zipf's distribution, as it is at least independent from the popularity of individual categories.

It can be shown that the above assumptions imply that the popularity of file k of type m is proportional to $1/(mk)$, which is the same as in [27].

3.2 Optimal caching performance

Having a well-defined semantic structure of the data, we now analyze the caching performance for a given static assignment of peers to super-peers. We aim to identify the optimal assignment of peer to super-peers, or in other words the optimal arrangement of pointers to super-peers in the super-peer caches. It is generally not obvious how to define the *optimality* of a particular setup. We describe the optimality in terms of performance and fairness stating that optimal means *Pareto optimal* [11]. An arrangement of items in super-peer caches is Pareto optimal if it is not possible to modify the contents of the cache of one peer in such a way that the fraction of requests produced by this peer that can be satisfied by the super-peers in its super-peer cache increases, while for all other peers this fraction does not decrease.

We can now prove that the semantic data model described in Section 3.1 has the following property.

Theorem A. *There exists an optimal arrangement of items in the super-peer caches such that the contents of the caches of all weak peers of the same semantic type are the same.*

PROOF. See the appendix.

This theorem can be shown to hold also in other models including the one defined in [27].

Theorem A can be used to prove the quality of the caching policies. The policy of the super-peer caches tends to group similar peers under the same super-peers. As a consequence, the caches of the weak peers of one type are very similar, and in an optimal situation they are the same. Furthermore, weak peers constantly look for super-peers that guarantee better hit ratios. Theorem A says that the system based on such relationships established between peer caches converges to an optimal setup.

3.3 Experimental setup

We simulate a system consisting of 100,000 peers, 100 super-peers, 10,000 files, and 20 semantic types. The value of the parameter α in Eq. 1 is set to 0.8. The size of the super-peer cache in the weak peers is set to 10, and the size of the file cache in the super-peers to 1,000. Before the simulation starts, all the super-peer caches have been filled with the identities of super-peers selected randomly and uniformly from the set of all super-peers. The file caches are initially empty.

Each peer stores 50 files selected randomly, taking into account the types of files and peers as explained in Section 3.1.

The super-peers are organized into a Gnutella-like network. Simple request flooding was employed for locating files which were not found in the local super-peer caches. Note that this choice does not influence our results.

The simulation is performed in phases. At the beginning of each phase we select randomly and uniformly one of the peers. This peer generates a search request. Additionally, in every 1,000,000th phase, each of the 100,000 peers sends information about one of the files which it stores locally to a super-peer selected randomly from its super-peer cache (the file insert protocol). Note that the situation in which file inserts are performed by all peers simultaneously is the worst possible. During the whole experiment, 10,000,000 phases are performed. The first 1,000,000 phases are treated as the bootstrap. The statistics are collected starting from phase 1,000,001 on.

For comparison, we measure the performance of one-level caching. This reference system does not make use of super-peers. Similarly as in [27], each peer has a semantic cache of peers that answered the peer's queries in the past. The caching policy used here is the same as the one deployed in the super-peer caches. The size of a peer's semantic cache is set to 300. Note that the total size of the caches used in the reference model, which is 30,000,000 (100,000 peers with caches of size 300 each) is almost thirty times higher than the total size of the caches used in our super-peer architecture, which is 1,100,000 (100,000 super-peer caches of size 10 and 100 file caches of size 1,000).

3.4 Results

This section presents the results of the experimental evaluation of the two-level caching architecture.

Figure 2 presents a comparison between the performance of our two-level caching architecture and the reference system which deploys one level of caches. For each file we compute the fraction of search requests targeting this file that is satisfied by one of the peer's direct neighbors. The direct neighbors in the reference model are the nodes stored in the peer's cache. In the two-level infrastructure the direct neighbors are defined as the super-peers contained in the super-peer cache. The files are sorted according to their semantic type. The set of 10,000 files is equally divided into 20 semantic types, with 500 files per type. Each file is assigned a unique identifier which we call its file rank — file k of type m has rank $(m-1)M+k$. The top left subplot of Figure 2 shows the hit ratios observed for our two-level caching architecture. The top right plot depicts results of the same experiment performed in a system with only one level of semantic caches. The bottom subplot presents the comparison of the efficiency of both caching schemes. It was obtained by applying curve fitting based on nonlinear regression [2] to the points showed in the above plots.

In most of the cases the super-peer algorithm outperforms the one-level design, even though the total number of items cached in the two-level system is

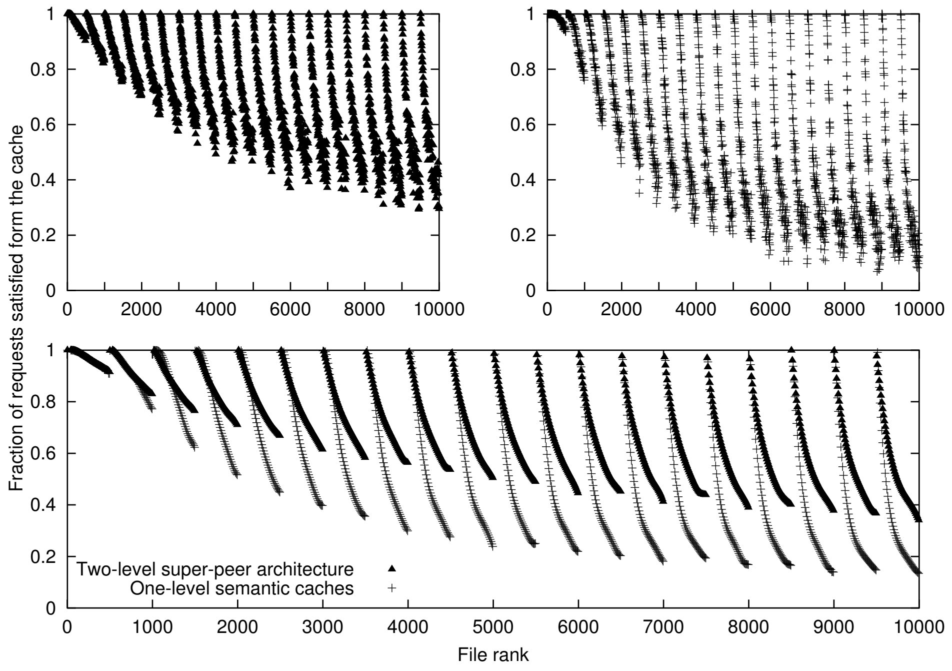

Fig. 2. The hit ratios of the two-level and one-level semantic caches.

much lower than in the reference model. Only for a small number of the most popular files the two approaches achieve comparable results. The average cache hit ratio defined as the percentage of the requests that were satisfied from the cache during the whole simulation for our super-peer architecture was 71% against 56% for the one-level caching scheme.

In order to investigate the properties of the mixed caching method we repeat the series of experiments described in Section 3.3 for LRU and LFU policies applied to the file caches. The results of the comparison are presented in Figure 3. The plot is divided into two subplots. The top subplot of Figure 3 shows the statistics for the files in the ten most popular categories, while the bottom subplot presents the hit ratios for the rest of the files. Also in this case we performed curve fitting to improve the clarity of the plot.

The LFU strategy promotes the most popular files, performing much worse that the other two strategies for unpopular files. The mixed method, while still providing lower cache miss rates for the popular items, performs much better for the unpopular content. We computed also the average cache hit ratios for LFU, LRU, and the mixed strategy, which are 62%, 63%, and 71%, respectively.

According to the protocol presented in Section 2.3, if the requested file is not cached by one of the super-peers known to the request initiator, a search in the underlying super-peer network is performed. It is possible that some of the

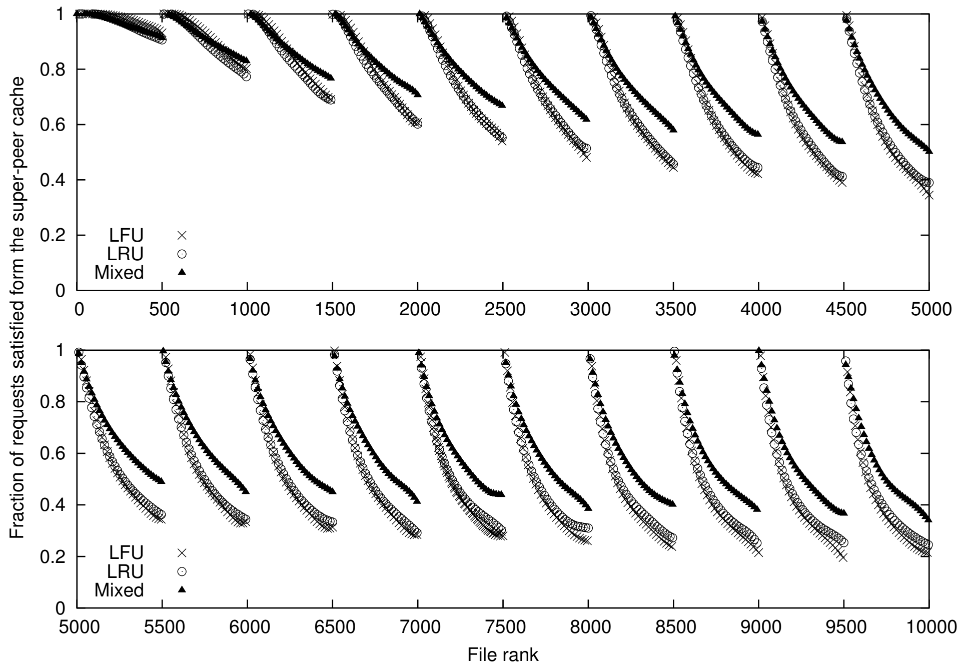

Fig. 3. A comparison of different caching strategies applied to file caches.

files possessed by the weak peers are not cached by any of the super-peers, and as a consequence, cannot be found during the super-peer search. We investigate the influence of different file cache management strategies on the number of files that are not cached by any super-peer. Figure 4 shows the fraction of requests addressing files of a particular type that could not have been satisfied because the located file was not indexed by any super-peer (the smaller the better).

We can clearly see that the mixed strategy in most of the cases outperforms the other two. The strong preference of the most popular files exhibited by the LFU policy is visible in the statistics for types 5 and 6. LRU is more fair than LFU for less popular files, but still falls behind the mixed strategy.

4 Conclusions and future work

Semantic caches as well as super-peers are meant to improve the performance and scalability of unstructured, symmetric P2P network designs. We showed that both these mechanisms can be integrated in a very natural way resulting in much higher performance than when deployed independently of each other.

The semantic caching model for a content sharing P2P network proposed in this paper is based on, and derives its advantages from three key issues. First, we

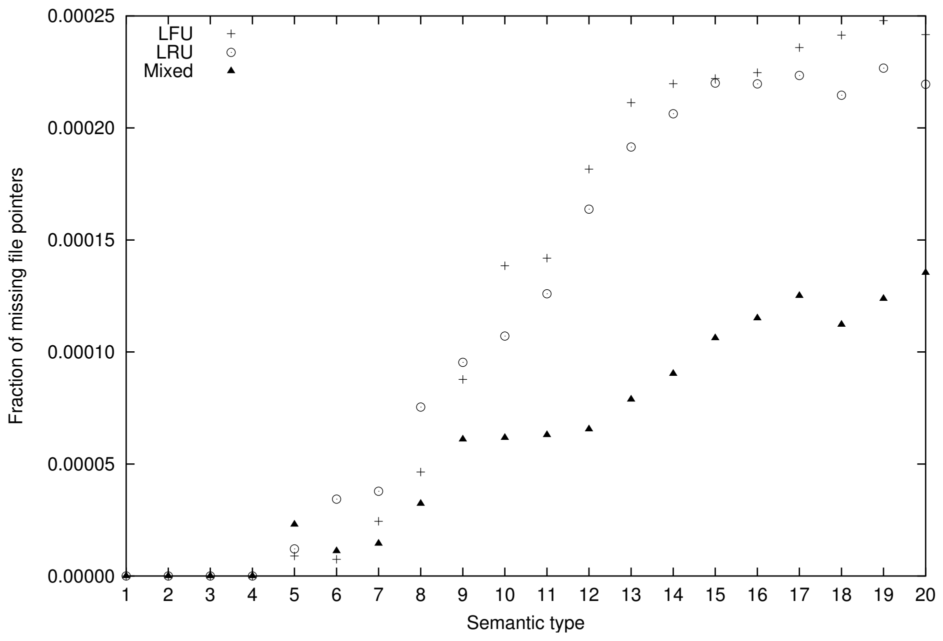

Fig. 4. The file coverage of the super-peer caches.

use the heterogeneity of the nodes by placing the semantic caches of file pointers only at the super-peers — selected nodes with higher capacities. These file caches are used by multiple peers at the same time, which shortens the cache warm-up period and increases the utilization of the cache. Second, weak peers maintain caches of super-peers which are most likely to know the location of the files they are interested in. Third, we introduce the mixed cache management policy for the file caches that achieves much higher cache hit ratios for less popular files being still very efficient for the most popular items. Furthermore, the probability that a file is not known to any super-peer in the network is much lower for the mixed policy than for LRU or LFU.

The analysis and the experimental evaluation on the artificially created data set give us an idea about the performance gain of using the proposed infrastructure. However, in order to determine how a super-peer architecture based on semantic caches performs in a real environment, more elaborate experiments are needed. For this purpose, we plan to use 2-year traces of the `suprnova.org` website which contain popularity statistics of the content distributed in the BitTorrent P2P network [20].

We consider also extensions of the cache adaptation algorithms. Interesting results may be achieved by allowing peers to adapt not only content but also sizes of the caches.

References

1. http://gnutella.wego.com.
2. http://www.curvefitting.com.
3. http://www.kazaa.com.
4. http://www.thepiratebay.org.
5. Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and zipf-like distributions: Evidence and implications. In *INFOCOM*, pages 126–134, March 1999.
6. Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, and Scott Shenker. Making gnutella-like p2p systems scalable. In *Proceedings of the ACM SIGCOMM 2003 Conference*, Karlsruhe, Germany, August 2003.
7. Arturo Crespo and Hector Garcia-Molina. Routing indices for peer-to-peer systems. In *Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, Vienna, Austria, July 2002.
8. F. Le Fessant, S. Handurukande, A.-M. Kermarrec, and L. Massoulie. Clustering in peer-to-peer file sharing workloads. In *3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, San Diego, CA, February 2004.
9. A.J. Ganesh, A.-M. Kermarrec, and L. Massouli. Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers*, 52(2), February 2003.
10. S. Handurukande, A.-M. Kermarrec, F. Le Fessant, and L. Massoulie. Exploiting semantic clustering in the edonkey p2p network. In *11th ACM SIGOPS European Workshop (SIGOPS)*, Leuven, Belgium, September 2004.
11. Marcus Hutter. Self-optimizing and Pareto-optimal policies in general environments based on Bayes-mixtures. In *Proceedings of the 15th Annual Conference on Computational Learning Theory (COLT 2002)*, Lecture Notes in Artificial Intelligence. Springer, 2002.
12. H.T. Kung and C. H. Wu. *Content Networks: Taxonomy and New Approaches.* Oxford University Press, 2002. to appear in The Internet as a Large-Scale Complex System.
13. N. Laoutaris, S. Syntila, and I. Stavrakakis. Meta algorithms for hierarchical web caches. In *IEEE International Performance Computing and Communications Conference (IEEE IPCCC)*, Phoenix, Arizona, April 2004.
14. X. Li and J. Wu. *Handbook of Theoretical and Algorithmic Aspects of Ad Hoc, Sensor, and Peer-to-Peer Networks*, chapter Searching Techniques in Peer-to-Peer Networks. CRC Press, 2005.
15. Alexander Lser, Martin Wolpers, Wolf Siberski, and Wolfgang Nejdl. Semantic overlay clusters within super-peer networks. In *International Workshop on Databases, Information Systems, and P2P Computing, colocated with 29th International Conference on Very Large Databases (VLDB2003)*, Berlin, Germany, september 2003.
16. Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. Peer-to-peer computing, 2002.
17. Wolfgang Nejdl, Wolf Siberski, Martin Wolpers, and Christoph Schmitz. Routing and clustering in schema-based super peer networks, 2002.
18. Wolfgang Nejdl, Martin Wolpers, Wolf Siberski, Christoph Schmitz, Mario Schlosser, Ingo Brunkhorst, and Alexander Lser. Super-peer-based routing strategies for rdf-based peer-to-peer networks. In *12th International World Wide Web Conference (WWW2003)*, Budapest, Hungary, may 2003.

19. A. Parker. The true picture of peer-to-peer filesharing, 2004. http://www.cachelogic.com/.
20. J.A. Pouwelse, P. Garbacki, D.H.J. Epema, and H.J. Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In *Proceedings of the 4th International Workshop on Peer-To-Peer Systems (IPTPS'05)*, Ithaca, New York, february 2005.
21. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. In *Proceedings of ACM SIGCOMM 2001*, 2001.
22. Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.
23. Stefan Saroiu, Krishna P. Gummadi, Richard J. Dunn, Steven D. Gribble, and Henry M. Levy. An analysis of internet content delivery systems. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, December 2002.
24. K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient content location using interest-based locality in peer-topeer systems. In *INFOCOM*, 2003.
25. Ion Stoica, Robert Morris, David Karger, and M. Francs Kaashoekand Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.
26. Sarut Vanichpun and Armand M. Makowski. The output of a cache under the independent reference model — where did the locality of reference go? In *SIGMETRICS 2004/PERFORMANCE 2004: Proceedings of the joint international conference on Measurement and modeling of computer systems*, pages 295–306, New York, 2004. ACM Press.
27. S. Voulgaris, A.-M. Kermarrec, L. Massoulie, and M. van Steen. Exploiting semantic proximity in peer-to-peer content searching. In *10th IEEE Int'l Workshop on Future Trends in Distributed Computing Systems (FTDCS 2004)*, Suzhou, China, may 2004.
28. Beverly Yang and Hector Garcia-Molina. Designing a super-peer network, 2003.
29. Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, January 2004.

Appendix

PROOF OF THEOREM A. Consider the file cache $s.F$ of super-peer s, and denote by $r_n(s)$ the fraction of all requests submitted to s that are issued by peers of type n. Assuming the Independent Reference Model [26] of requests and a reasonable caching policy at the file caches, the cache hit ratio of a peer of type n is non-decreasing function of $r_n(s)$ — the more requests are produced by peers of type n, the better cache $s.F$ adapts to the needs of peers of this type.

We say that an arrangement of items in the super-peer caches is *structured* if the caches of all peers of the same semantic type are the same.

Consider a non-optimal structured arrangement of items in the super-peer caches. Non-optimal means (see Section 3.2) that it is possible to modify a super-peer cache of one of the peers, say p, in such a way that two conditions hold.

First, the fraction of requests produced by p that can be satisfied by the super-peers in p's super-peer cache increases. Second, for all other peers this fraction does not decrease.

A cache modification for a peer p of type n influences the values of the $r_n(s)$ of some of the file caches at the super-peers. Removing any super-peer s from p's cache decreases (or does not influence) $r_n(s)$, and increases (or does not influence) $r_m(s)$, where n is the type of p and m is different than n. Subsequently adding any super-peer s to p's cache has the opposite effect — $r_n(s)$ increases (or does not change), and $r_m(s)$ decreases (or does not change). Furthermore, if $r_l(s)$ increased (decreased) after modifying p's cache, then it will also increase (decrease) when we perform the same modification to the cache in p', where p' is a peer of the same type as p. We showed earlier that the cache hit ratio of a peer of type l at super-peer s is a monotonic function of $r_l(s)$. We conclude that if a certain modification in p's cache improves the hit ratios at some peers and does not decrease the hit ratios at all other peers, then applying the same modification to the cache in p' will improve the same hit ratios, and will not decrease the other.

We have just shown that if a certain modification of a super-peer cache of peer p improves the non-optimal, structured arrangement of super-peer caches, then by applying the same modification to the super-peer caches of all peers of the same type as p the arrangement can be improved by at least the same amount. Such a modification results in an arrangement that is again structured. There is, however, a finite number of (structured) arrangements, which means that we cannot endlessly improve. At some point we will end up with a structured arrangement which is optimal.