Self-Organizing Peers for Electing Super-Peers in Peer-to-Peer Networks

R.Venkadeshan
Assistant Professor/CSE Department,
Chettinad College of Engg & Tech,
Karur, Tamilnadu, India,

Dr.M.Chandrasekar
Professor / CSE Department
VKS College of Engg & Tech
Karur, Tamilnadu, India,

M.Jegatha
Assistant Professor/CSE Department,
Chettinad College of Engg & Tech,
Karur, Tamilnadu, India,

Abstract: Peer-to-Peer (P2P) networks provide a significant solution for file sharing among peers connected to Internet. It is fast and completely decentralized system with robustness. But due to absence of a server documents on a P2P network are not rated which makes it difficult for a peer to obtain precise information in result of a query. Two-layer hierarchy unstructured peer-to-peer (P2P) systems, comprising an upper layer of super-peers and an underlying layer of ordinary peers, are commonly used to improve the performance of large-scale P2P systems. Super-peer architectures exploit the heterogeneity of nodes in a P2P network by assigning additional responsibilities to higher-capacity nodes. In the design of a super-peer network for file sharing, several issues have to be addressed: how client peers are related to super-peers, how super-peers locate files, how the load is balanced among the super-peers, and how the system deals with node failures. In this paper we introduce the super-peer network architecture (SPNA) that solves these issues in a fully decentralized manner. SPNA maintains a super-peer network topology that reflects the semantic similarity of peers sharing content interests. Super-peers maintain semantic caches of pointers to files which are requested by peers with similar interests. Ordinary peers, on the other hand, dynamically select super-peers offering the best search performance. On the face of it, super-peers bear a much larger traffic burden than normal peers and receive negligible improvement in search performance when compared to any of the nodes they serve.

Keywords: Unstructured Peer-to-Peer Networks, Super Peers Networks (SPN) Architecture.

1. INTRODUCTION

Unstructured peer-to-peer networks like Gnutella and KaZaA are characterized by the absence of specific mechanisms for enforcing a particular network topology or file placement. The design of search algorithms is critical to the performance of unstructured peer-to-peer (P2P) networks. In the unstructured P2P networks, each node does not have the global information about the whole topology and the location of queried resources.

As a result, search proceeds by flooding queries to all nodes within a certain search horizon. Researchers have recently proposed extensions to the flooding mechanism, such as expanding ring search and random walks, that can improve search performance [1]. Unfortunately, these extensions require modifications to both the software and the protocols used at every node in the network. In contrast, systems like KaZaA and more recent versions of Gnutella improve both the efficiency and effectiveness of the search process by introducing a notion of network structure, elevating certain well provisioned nodes to the role of super-peers. Super-peers serve as network hubs that index files belonging to other nodes. The resulting architectures break the symmetry of pure P2P systems by assigning additional responsibilities to high-capacity nodes called *super-peers*. In a super-peer network, a super-peer acts as a server to *client (ordinary, weak) peers*. Weak peers submit queries to their super-peers and receive results from them. Super-peers are connected to each other by an overlay network of their own, submitting and answering requests on behalf of the weak peers.

It was observed that by grouping peers interested in similar files and routing their search requests within these groups that the performance of locating content can be greatly improved. The biggest challenge is, thus, to build an architecture that maintains and exploits the discovered semantic structure. In this paper we present the design and evaluation of a P2P architecture that combines the homogeneity of peer interests with the heterogeneity of peer capacities to solve the problem of efficient peer relationship management. The design of our self-organizing super-peer network is guided by the following set of requirements. The self-organization property of Super Peer Network, requires that the system is able to discover and exploit the semantic structure present in the network no matter what the initial topology is. A new peer joining the network has no knowledge about the system and is connected to a set of randomly selected nodes. The longer a peer stays in the

system, the more information it can collect and exploit for improving the performance of its searches [2]. The time that it takes a new peer to achieve its optimal performance should be minimized. SPNA uses two-level semantic caches deployed at both the super-peer and the weak-peer level to maintain relationships between related peers and files. The cache maintained by a super-peer contains references to those files which were recently requested by its weak peers, while the cache of a weak peer stores references to those super-peers that satisfied most of its requests. First, we improve the algorithm for establishing relationships among peers by allowing weak peers to contact each other directly and exchange information on other peers in the network. Second, we present a mechanism for balancing the load between super-peers. Finally, measure the time needed to ¯nd an optimal set of neighbors for each peer, which all helps to understand how the system would perform in a real environment. Therefore, super-peer overlay networks offer the potential for building efficient and scalable file-sharing systems. However, establishing the optimal super-peer network design necessarily involves making various performance tradeoffs and raises a number of key questions. For example, how should the super peers connect with one another? How should a suitable topology be chosen for the super-peer overlay network? How should the network design utilize an efficient broadcasting algorithm to avoid broadcast storms and redundant messages? To what extent does the design provide a reliable service given the possibility that a hierarchical super-peer represents a potential point of failure for multiple associated clients?

2. SELF-ORGANIZING SUPER PEER NETWORKS

The relationships are organized by defining for each peer the set of other peers, called the *neighbors*, it interacts with. In super-peer networks such as Kazaa, Gnutella ultrapeers, and Chord super-peers, neighbors are selected from the set of high-capacity peers called super-peers; low-capacity peers - the week peers - cannot become neighbors.

2.1 Architecture Overview

The basic idea behind the system architecture proposed by us is simple and intuitive. Weak peers with similar interests are connected to the same super-peers. The request locality suggests the usage of caches that store the results of recent searches. But not only super-peers are responsible for discovering semantic structure in the network. We also allow weak peers to collect statistics about the content indexed by the super-peers. Having this information, weak peers can make local decisions about which super-peers to connect to. In our architecture, super-peers store the information about the location of the content recently requested by their weak peers. Weak peers, on the other hand, sort the super-peers known to them according to the number of positive responses to their queries, and prefer to connect to super-peers that have satisfied most of their requests. To accelerate the process of grouping peers with similar interests under the same super-peers, we allow weak peers to exchange their lists of super-peers. More precisely, if a search succeeds, the requesting peer asks the peer that has the requested file for its list of top-ranked super-peers. This list is then merged with the list of super-peers known to the requesting peer. The intuition here is that if both peers were interested in the same file, then it is highly probable that they will share interest for more files in the future.

2.2 System Model

The information stored at a node in our system depends on the type of this node. Each weak peer maintains a *super-peer cache* which contains the identities of super-peers (e.g., their IP addresses and port numbers). Each super-peer has a *file cache* of pointers to files stored at the peers. The relationships between SPNA peers are presented in Figure 1. 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*. 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. 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*.

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. The super-peer and file caches are controlled according to different caching policies. The priority of a super-peer in a super-peer cache is increased by one after every positive feedback provided by this super-peer. The priority of a file pointer in a file cache is modified according to the *mixed* policy. If the file pointer was already in the cache then the corresponding priority is increased by one. Otherwise, the file pointer is added to the cache with its priority one higher than the highest priority of all other cached items. The super-peers in our system index individually selected files rather than the entire sets of files stored at their weak peers. This type of architecture can deal with a situation in which a single weak peer has files of different semantic types. Pointers to these files can then be cached by different super-peers. Additionally, we require that the probability that a

search succeeds is high when the requested information is present at least at one of the super-peers.

directing the lookup messages issued by the ordinary peers towards the high-capacity nodes in the system.

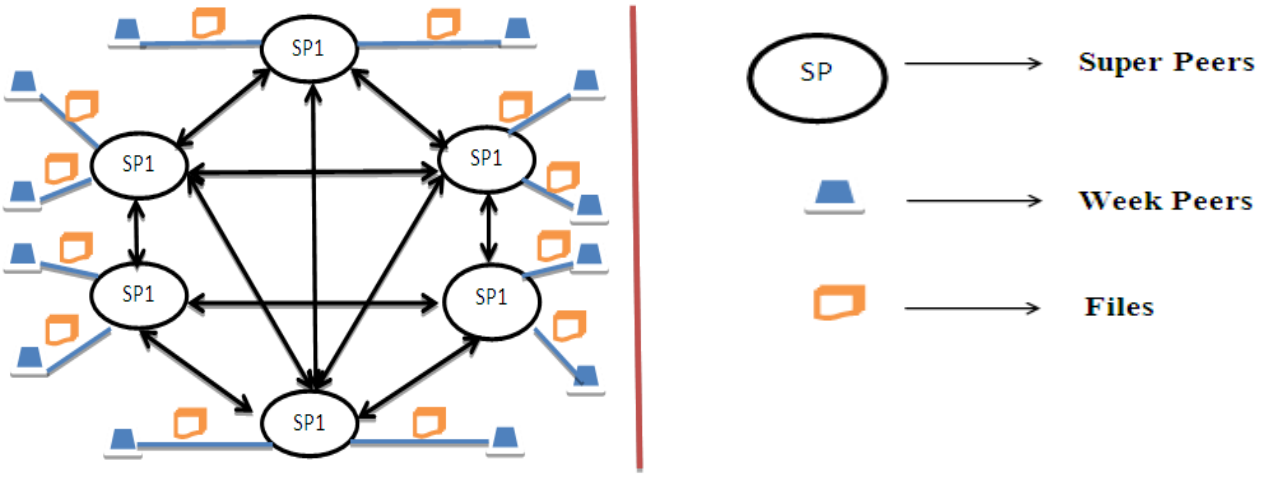

Fig. 1. Architecture of SPN

3. RELATED WORKS

In recent years, various hierarchical two-layer unstructured P2P systems have been proposed as a means of scaling up conventional unstructured P2P systems. Such systems, of which Gnutella vs. 6 [12] and KaZaA [13] are the most widely used, comprise super-peers and ordinary peers and have a number of key advantages for the execution of large-scale distributed applications, including a higher search efficiency and the ability to harness the power and resources of multiple heterogeneous nodes. However, they also suffer the problems of a heavy workload and the risk of single-point failures, i.e. the failure or departure of a single super-peer causes all of its children (ordinary peers) to lose their connections to the system until they are reassigned to a new super-peer.

In an attempt to address these issues, Yang *et al.* [21] proposed several rules of thumb for accomplishing the major trade-offs required in super-peer networks and introduced a k-redundancy concept for improving the system reliability and reducing the workload imposed on the individual super-peers. Watababe *et al.* [26] presented a method for reducing communication overheads in a two-layer hierarchical P2P system by allowing ordinary peers within designated clusters to communicate directly with one another rather than through a super-peer. Gia [27] improved the performance of unstructured P2P systems by using a dynamic scheme to select appropriate super-peers and to construct the topology around them in an adaptive manner. Furthermore, a search-based random walk mechanism was proposed for

However, the efficiency of the search procedure relies fundamentally on the matching data being found very quickly. In the worst case scenario, the random walk search mechanism either gives up without finding a match or may have to traverse a very long path. Pyun [28] presented a protocol designated as SUPs for constructing the super-peer overlay topology of scalable unstructured P2P systems using a random graph method.

The results showed that SUPs was not only more computationally straightforward than the scheme presented in [34], but also was much compatible with existing system and was likely be adopted. Although the resulting overlay network had a lower diameter and the topologies produced were low cost and almost regular, the authors didn't discuss the content search procedure in detail.

Xiao *et al.* [10] presented a workload model for establishing the optimal size ratio between the super-layer and the leaf-layer, and proposed an efficient dynamic layer management (DLM) scheme for super-peer architectures. In the proposed approach, the DLM algorithm automatically selects the peers with larger lifetimes and capacities as super-peers and designates those with shorter lifetimes and capacities as leaf peers. However, the DLM algorithm inevitably incurs a substantial traffic overhead in exchanging information amongst neighboring peers and a peer adjustment overhead is incurred when a super-peer is demoted to be a leaf-peer. Moreover, they did not examine which topology is suitable for super-peers to

maximize their benefits.

4. SUPER PEER NETWORKS SEARCH PROTOCOLS

Peers use the information collected during past searches to improve the performance of future requests. The contents of the super-peer and file caches are reorganized depending on the feedback provided by peers involved in the search process. The pseudo-code of the search algorithm employed in our self-organizing super-peer network presented in Figure 2 is divided into four subroutines. The super-peer cache of peer *p* is denoted by *p.S*, while the file cache of super-peer *s* is represented as *s.F* . The main search algorithm is the function peer search. When a weak peer *p* looks for a file *f*, it first checks the file caches of the super-peers known to it (line 2). Note that *p* starts with the super-peers with the highest priorities. When the file is found (line 4), a pointer to super-peer *s* that knows the location of *f* is stored for future reference (line 5).

However, if the file was not found with this method (line 7), the search request is forwarded to one of the super-peers in *p*'s super-peer cache selected according to a random distribution biased towards super-peers with higher priority (line 8). This super-peer is further responsible for locating file *f*. If the search succeeds, a pair < *q; t* >, where *q* is a peer that has *f* and *t* is a super-peer that has a pointer < *f; q* > in its file cache, is returned to *p* (line 9). At this point the self-(re)organization process begins. This process is performed in two stages. First, peer *p* increases the priority of the super-peer *t* that satisfied the search request (lines 12|15). As a consequence, in the future *p* will direct more of its requests to *t*. Second, *p* integrates the list of super-peers kept by the weak peer *q* with its own super-peer cache (line 16).

We exploit here a simple, yet powerful principle called *interest-based locality*, which postulates that if *p* and *q* are interested in the same file, it is very likely that more of their requests will overlap. It is thus beneficial for both *p* and *q* to use the same set of super-peers. The algorithm of the **super_peer_local_search** is straightforward. The search succeeds only if a pointer to file *f* is present in the file cache of super-peer *s* (line 19). Before returning the peer *q* that possesses file *f* (line 21), the priority of the corresponding cache item is increased (line 20). The function **super-peer_search** performs the search in the super-peer network (line 25). Upon receipt of the search results, a pointer to the requested file *f* and to the peer *q* holding file *f* are added to the file cache of *s* (line 27). The return value of the function (line 28) contains not only the peer *q*, but also the super-peer *t* that has a pointer to *f* in its file cache. The last function presented in Figure 2, **merge_super-peer_caches**, takes two parameters representing two peers *p* and *q*. The super-peer cache of peer *p* is updated with the content of *q*'s super-peer cache (lines 32 and 33). The functionality of merging the super-peer caches is not crucial for the system operation, but it accelerates the process of grouping weak peers under the same super-peers which improves the search performance.

```
1 peer_search(p : peer, f : file_name):
2   for s in p.S ordered according to decreasing
    priorities do
3   q← super-peer local search(s,f)
4   if super-peer local search succeeded then
5     t ← s
6     break
7   if f was not found until now then
8     s ←super-peer in p.S selected randomly with
      probability proportional to its priority in p:S
9     < q; t > ←super-peer search(s,f)
10    if super-peer search did not succeed then
11      return ERROR "File f not found"
12  if p.S contains t then
13    increase the priority of t in p.S
14 else
15    insert t into p.S
16  merge super-peer caches (p, q):
17  return q

18 super-peer_local_search(s:super-peer,f: filename):
19 if an entry < f; q > exists in cache s.F then
20    increase the priority of < f; q > in s.F
21    return q
22 else
23    return ERROR "File f not found"

24 super-peer_search(s, f):
25   perform a search in the super-peer network to
     locate a super-peer t which has an entry < f; q >
     in its cache
26   if search succeeded then
27     insert < f; q > into s.F
28     return < q; t >
29   else
30     return ERROR "File f not found"

31 merge_super-peer_caches(p : peer, q : peer):
32   for s in q.S do
33     if p.S contains s then
34       increase priority of s in p.S
35     else
36       insert s into p.S
```

Fig. 2. Pseudocode of Search Protocol in SPNA

4.1. Topic Based Search Optimization

Mere participation in a peer-to-peer protocol by deploying a super-peer infrastructure is likely not sufficient to exert any control over client behavior. To fully realize the benefits of being able to control and manipulate the peer-to-peer protocol, it is essential for the super peers to ensure that

a large fraction of the queries are routed through the super-peer infrastructure. It follows that some incentive mechanism is required to encourage individual peers to preferentially connect through these super-peers. By giving end users a superior search experience, both in terms of the ability to find what is being searched for as well as faster downloads, super peers
provide a strong incentive for end users to remain connected to them. Newer versions of some peer-to peer applications give the end user the ability to choose to which super-peer its client software connects [3]. Alternatively, the application logic itself may make this decision by choosing peers that return better results. In either case, the superior quality and performance of searches increases the likelihood that individual peers will connect to a super-peer infrastructure in preference to other peers in the system. Super-peers therefore require specific mechanisms to improve search performance for end-user queries. Accordingly, we introduce a new paradigm for query routing in our super-peers, which we call *topic-based search optimization*. Our strategy tries to exploit the following observations:

- There is likely to be significant locality in the type of content requested by individual peers, *i.e.*, peers are likely to be able to respond to queries similar to queries they themselves have made in the past.
- Super-peers are in a unique position to generate aggregated views of the content in the network by observing the queries that are routed through them.

Topic-based search optimization involves generating a profile of the content currently available in the network by categorizing it into distinct topics. The topics are created dynamically by analyzing the meta data contained in Query Response messages routed via the super-peer. Since most queries are routed through a super-peer, it is in a unique position to be able to generate this topic based view of the content and its location in the network. Meta data associated with an object typically contains a set of key-value pairs pertinent to the type of the object. For example, in the case of an audio file the meta data may contain one or more values in the following fields: *title, artist, album, category, release year, bit rate, length, description*, and *keywords*. In addition to meta data the Query Response message also provides a unique *signature* for every item returned in the form of a SHA digest.

Figure 3 shows meta-data information gathered from real Query Hit responses, and Figure 2 shows a view of this information in a multi-dimensional name space. Topics can be created by considering the type of objects, the category, the artist, or any representative combination of the fields that can be gathered from the meta data. Associated with each topic is a list of peers who are likely to be interested in that particular topic based on their past query history. This list provides the super peer with a wide selection of candidate peers who are likely to have related content. Topic-based search optimization improves upon two simpler potential strategies for improving search performance: The content itself could be cached at the super-peer, requiring large amounts of space and possibly incurring legal liability. Alternatively, the super-peer could maintain an index for each individual piece of content, rather than aggregating based on topics. We believe that, given the huge amounts of content available as well as the large number of peers we expect to connect to a single super-peer (a few thousand in our traces), these strategies are inefficient. Furthermore, we believe they might not even be necessary given sufficient effectiveness of topic-based search optimization. Also, unlike several previous schemes that exploit the locality of client interests, topic-based search optimization operates transparently to the end user.

4.2 Balancing the load among Super-Peers

Load balancing is critical to the availability, accessibility, scalability, and throughput of a P2P system. Poor load balancing may gradually transform the super-peer network into a backbone network as was observed for Gnutella [11]. The idea here is to avoid overloading individual super-peers, which is the case when some super-peers are getting significantly more queries than others. Before describing the load-balancing mechanism of SPNet, we first define the requirements of load balancing for a super-peer network in general. A minimal requirement is to prevent situations in which the load imposed on some super-peers exceeds their capacity limits. A more advanced load-balancing solution can further guarantee that the load assigned to each super-peer is proportional to its capacity. Finally, the performance overhead and implementation burden incurred by adding the load-balancing extensions should be low. In the remainder of this section we show how the above goals can be easily achieved by exploiting the properties of the self-organizing super-peer network.

At first sight the load-balancing problem that we face in the SPNet design seems to be more difficult than in other super-peer networks because the SPNet super-peers do not explicitly know their weak peers. Furthermore, in the SPNet architecture, the assignment of weak peers to super-peers is not fixed. As a consequence, the super-peers cannot transfer the weak peers between each other without the active Cooperation of the weak-peer layer. Being aware of these limitations, we have built into the search protocol a mechanism that indirectly influences the set of super-peers contacted by the weak peers by discouraging directing requests to overloaded super-peers. The basic idea behind the load-balancing mechanism of SPNet relies on the observation that a super-peer may control the number of received requests by affecting its priority in the super-peer caches of weak peers. An overloaded super-peer can simply start dropping some of the requests, effectively decreasing its priority in the

super-peer caches of the requesting peers. As the priority of a super-peer has a direct impact on the probability of that super-peer being selected as a request target, the load exercised on the overloaded super-peer will gradually decrease. Note that if a super-peer *s* refuses to service a request then eventually the client peer will ask another super-peer *t* to search for the file and to subsequently store a reference in its file cache. In other words, *t* will eventually take over some of the file references that were cached by *s*. The requirement that the load experienced by a super-peer is proportional to its capacity involves relating the effective load of that super-peer to the loads of other super-peers in the system. To avoid introducing an independent load-information exchange protocol, we let super-peers gather load values of other nodes while performing the search. In one specific case the behavior of the load-balancing algorithm can be confusing. Let's assume that super-peer *s* is overloaded and that it has in its cache the pointer < *f; q* > to file *f* requested by *p*. The request will be forwarded to another super-peer, say *t*. Super-peer *t* will then perform a super-peer search, find *s*, store pointer to *f* in its own cache, but return < *q; s* > to *p*. As a consequence, peer *p* will increase the priority of *s* in its super-peer cache. This behavior is counterintuitive as *p* should be discouraged to contact *s* in the near future. However, the increase of the priority of *s* should be interpreted as a one-time tradeoff. If a different peer sends subsequently a request for file *f* to *t*, super-peer *t* will satisfy the request from its local file cache.

Our load-balancing algorithm has thus the highly desired property of replicating file pointers cached by the overloaded super-peers at lighter-loaded peers. The load-balancing scheme that we presented here is simple yet powerful and extremely flexible. While many state-of-the-art load-balancing algorithms assume that all peers have equal capacities, our self-organizing architecture can deal with arbitrary capacity values and even allows these values to be changed during system operation. The load imbalance caused by a change of the parameters of the super-peers is automatically taken into account, and the system gradually adapts to the new circumstances. Because neither the weak peers nor the file pointers have to be explicitly reassigned from one super-peer to another, no complex overlay infrastructure such as *virtual servers* or *buckets* of file identifiers needs to be introduced.

4.3 Ordinary Peer Entry in P2P Networks

1. When a node *x* enters the system, it immediately becomes its own group G_x. G_x forms a list of credentials, including its bandwidth capabilities and its number of public files.
2. Node *x* contacts a well-known location, called a "node cacher", to find other nodes *S* in the system (similar to Gnutella and Mojo Nation). This bootstrapping component only keeps a list of other nodes that have recently contacted it, also trying to find other nodes in the system. Because this list is the only state it contains, it can be easily replicated, and can pop in and out of existence.
3. Using the nodes *S* that the new node learns about from the "node cacher," G_x forwards its credentials to the groups containing $s \in S$, by sending messages to each _, which then forward this information to their roots.
4. Each root that considers G_x valid for entry respond to G_x with its credentials. G_x chooses which group to join by picking the one with the best credentials. If this group agrees, G_x then merges with this group. If it refuses, G_x tries another group.

4.4 Ordinary Peer Exit

We have experimented with two designs for keeping the descendants of a node part of a group when a node exits. In one mechanism, nodes try to maintain knowledge of their siblings and grandparents. This information is sufficient to elect a new leader to take the place of the missing parent and then contact the grandparent to inform it of the new topology, including the summary filter change. The other, lazy mechanism just drops children from a group when their parent dies. This expends fewer topology messages and is the one we use in the simulation.

5. EXPERIMENTAL SETUP

We evaluate our proposal through trace-based simulation. For the purposes of data collection we implemented a Gnutella client based on the publicly available Mutella [4] source code. When running, the client actively participates in the public Gnutella network as a super-peer. Our data-collection super-peers do not introduce any traffic into the network; they only gather the data that is being routed through them. Each super peer allows between 20 and 200 leaf-nodes to connect to it. We have not yet compared the results of our study— which uses version 0.6 of the Gnutella protocol [5]—to those of a previous study using an earlier version [6]. We are interested in comparing the effectiveness of our scheme against a naive caching approach that simply
stores a copy of all requested data.

Figure 3 presents the query activity of a randomly selected two-hour period from our trace files. The absolute number of queries per minute is labeled QR. As a base line, we simulate the optimal caching strategy by checking the signatures contained in all Query Responses against all previously received response signatures (recall that this is a two hour section from the middle of our trace, so the cache is already warm). Hence, the line SC represents an optimistic upper bound for the performance of proposed caching based schemes. It is problematic to compare the performance of our scheme, as we return related search results that might not have been returned in the original network. Instead, we attempt to provide a rough estimate of the likelihood that topic-based search would return useful responses. The curve

labeled TC represents the number of queries which had overlapping interest with at least one cached topic group.

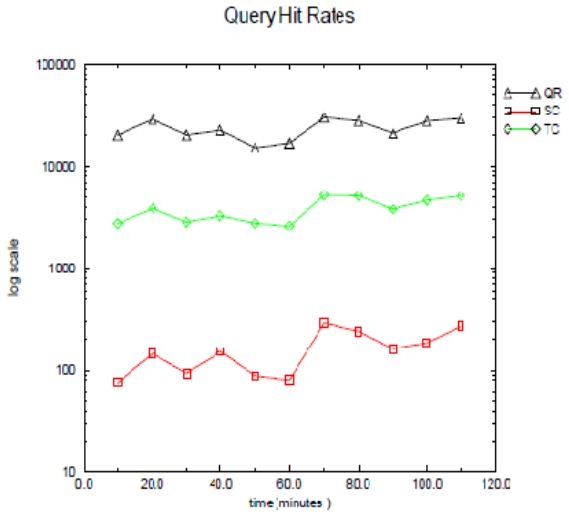

Fig. 3. The Curve QR represents the number of queries, the curve SC represents the cache hit rate for objects matching exactly, and the curve TC represents the hit rate for the queries matching a topic of interest

Although the number of queries that match this criteria is much higher than the number of queries that match the cached objects, we cannot directly compare the two. Recall the topic-based cache is built by aggregating the client's query requests and responses— thus we have no way to know what they actually store. Therefore there are no guarantees that clients will find the data they are looking for in any of the peers querying for content related to the topic. The likelihood of success is tied directly to the locality of clients' interest; that is, the chance that clients actually share data related to the topics of their queries. We quantify the locality of client interest using a,

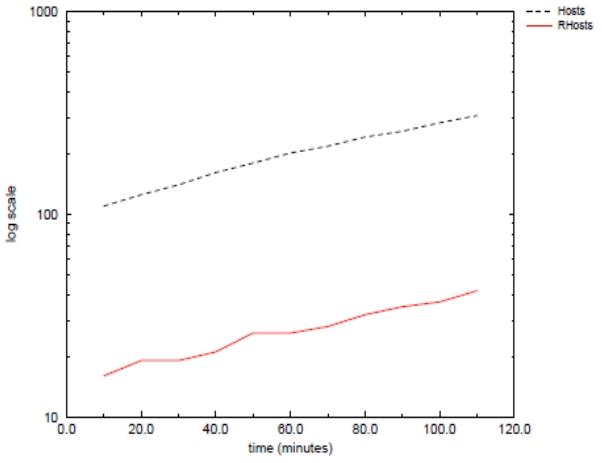

Fig. 4. This graphs compares the number of neighboring hosts making queries (Hosts), to the number of these hosts (RHosts) responding with useful results when sent a query

separate study. We wrote a second crawler based on the Gnutella protocol that connects to a large number of peers. Upon receiving a query from a neighboring peer, it extracts the first two words from the query, and creates a new query from these words and sends them to the neighbor the query had originated at. Even though this does not precisely capture the users interest topic, as many times the first two words signify nothing in particular, we believe they represent a crude notion of a search topic. The results of the crawler represented in Figure 4 seem promising: It is clear from the figure that approximately 15% of the peers do indeed respond to the queries that we are sending to them.

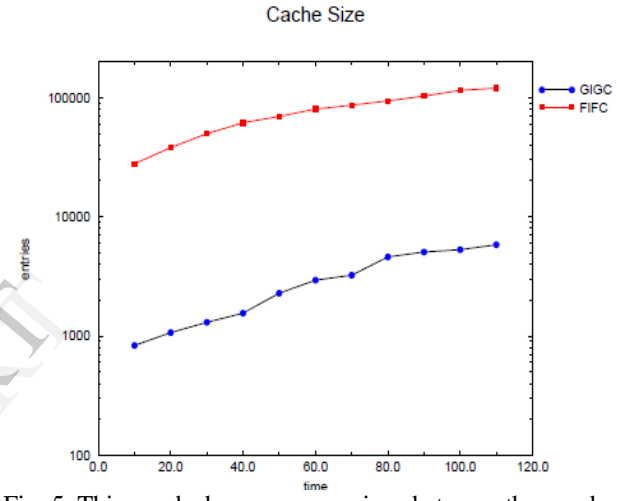

Fig. 5. This graph shows a comparison between the number of active groups in the topic-based cache (GIGC), and the expected number of files in the file cache (FIFC).

Combing the results of the previous two studies, we can deduce a very rough approximation of the effectiveness of topic-based search optimization. If we consider the 15% hit rate from above as representative of the response rate for peers in a given topic then we can infer from Figure 3 that topic-based search optimization is likely to out perform an infinitely large cache, since 15% of the TC curve is still substantially above the SC line. Furthermore, the storage requirements are substantially less. Figure 5 shows the number cached objects in our simulated cache as compared to the number of groups. Keeping in mind that the average size of each object is over 5 Megabytes, it's clear that topic-based search optimization is far more practical to implement.

A major concern with our approach, and, indeed any approach that only caches pointers to peers storing data as opposed to the data itself, is that the peers will not be available when subsequent queries arrive. Figure 6 attempts to quell that fear by plotting the uptime of all peers that

connected to our super-peers during the duration of the trace. Note that a large number of peers remain connected for a substantial period of time.

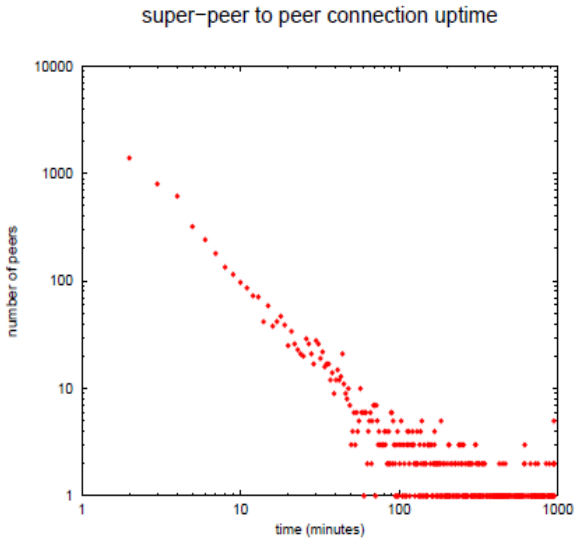

Fig. 6. The duration of peer relationships

6. CONCLUSION

We have introduced a self-organizing super-peer network architecture called SPANet built on top of an unstructured topology with semantic correlations between peers and files. Starting with random sets of neighbors, peers are always able to find super-peers which guarantee the highest performance of their searches. All decisions in our system are made locally by each peer based on the information collected during previous searches. We have also proposed a novel performance model of a P2P network where peer requests exhibit semantic patterns. Through simulations with real-world trace-based data, we have shown that in SPANet not only very popular files, but also less popular content can be located very efficiently. Further, we have demonstrated that a new peer that joins the system can very quickly find the set of super-peers that guarantee the highest performance. To that end, we described a novel mechanism called topic based search that improves search performance for client nodes by caching meta data at super-peers. Exploiting locality in user interests to improve search is a promising approach that has been pursued previously. Our approach is unique, however, in that information is collated at the super-nodes transparently to the end users.

REFERENCES

[1] Sabu, M.T. and S.K. Chandra, 2010. Survey of search and replication schemes in unstructured P2P networks. Network Protocols Algorithms, 2: 93-131.

[2] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *Proc. 16th Annual ACM Int'l Conference on Supercomputing*, 2002.

[3] S. Sen and J. Wang, "Analyzing peer-to-peer traffic across large networks," in *Proc. Second Annual ACM Internet Measurement Workshop*, Nov. 2002, pp. 137–150.

[4] A. Gerber, J. Houle, H. Nguyen, M. Roughan, and S. Sen, "P2P: the gorilla in the cable," in *Proc. National Cable & Telecommunications Association (NCTA)*, June 2003, (to appear).

[5] "Mutella," 2003. [Online]. Available: mutella. sourceforge.net

[6] "Gnutella v0.6," 2002. [Online]. Available: groups. yahoo.com/groups/files/Development/ Gnutella Protocolv 0.6- 200206draft.txt

[7] M. Ripeanu, I. Foster, and A. Iamnitchi, "Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design," *IEEE Internet Computing Journal*, vol. 6, no. 1, 2002.

[8] E. Cohen and S. Shenker, "Replication strategies in unstructured peer-to-peer networks," in *Proc. ACM SIGCOMM*, Oct. 2002, pp. 177–190.

[9] "Sandvine." [Online]. Available: www.sandvine.com/

[10] E. Cohen, A. Fiat, and H. Kaplan, "Associative search in peer-to-peer networks: Harnessing latent semantics," in *Proc. IEEE INFOCOM*, Apr. 2003.

[11] K. Sripanidkulchai, B. Maggs, and H. Zhang, "Enabling efficient content location and retrieval in peer-to-peer systems by exploiting locality in interests," *ACM Computer Communication Review*, vol. 32, Jan. 2002.

[12] Gnutella - A protocol for Revolution, http://rfcgnutella. sourceforge.net.com/.

[13] KaZaA available at http://www.kazaa.com/. http://www.

[14] Overnet/edonkey2000, available at . edonkey2000.com/, 2000.

[15] Bittorrent, available at http://bitconjurer.org/BitTorrent/, 2003.

[16] Saeyoung Han and Sungyong Park, "A Dynamic Layer Management Scheme for a Superpeer Ring with a Loosely Consistent DHT", IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, pp. 383-386, Aug. 2007.

[17] Yingwu Zhu and Yiming Hu. Efficient, proximity-aware load balancing for dht-based p2p systems. *IEEE Transactions on Parallel and Distributed Systems*, 16(4):349 { 361, April 2005.

[18] David R. Karger and Matthias Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *SPAA '04: Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and*

Architectures, pages 36{43, New York, NY, 2004.
ACM Press.