

Scalable Peer-to-Peer File Sharing with Efficient Complex Query Support

Yan Li¹, Jyoti Ahuja², Li Lao³, Jun-Hong Cui¹

yan.li@uconn.edu, jyoti.ahuja@gmail.com, llao@google.com, jcui@cse.uconn.edu

¹ Computer Science & Engineering, University of Connecticut, Storrs, CT 06269

² Yahoo Software Development India Pvt Ltd, Bangalore, India 560001

³ Google Santa Monica, 604 Arizona Avenue, Santa Monica, CA 90401

Abstract—A good P2P file sharing system is usually expected to achieve the following design goals: scalability, routing efficiency and complex query support. In this paper, we propose such a system, called PSON, which can satisfy all the three requirements. PSON is essentially a semantic overlay network of logical nodes, in which queries are routed on the basis of semantics. A logical node is formed by a cluster of peers that are close to each other in the physical network. Each cluster selects a powerful peer as super peer to support routing in the overlay network. To facilitate routing, all the super peers (or logical nodes) are organized in the form of a balanced tree. By exploiting the concepts of hierarchy and semantics, PSON can support complex queries in a scalable and efficient way. In this paper, we will describe the system architecture, and examine the key component of PSON design, i.e., semantic overlay construction and routing. We also conduct simulations, and show that the query routing in PSON is very efficient ($O(\log(n))$ in the case of exact query).

I. INTRODUCTION

A successful P2P file sharing system should achieve the following design goals. First, the system should scale to large numbers of peers spreading throughout a wide network across different administrative domains. Second, the system needs to provide an efficient and effective file lookup (or search) scheme. It should return the location (e.g., the IP addresses of the peers who have the file) of a requested file with minimal communication and computation overhead. Third, the system should support complex (or partial-match) queries. Shared files have attributes that describe their properties, e.g., singer, composer and title for a music file. It is desired that the search mechanism could support partial match queries that contain a subset of the attributes (e.g., “search all rock music files composed in year 2003”) and may even contain typos.

In the literature, there are many P2P files sharing systems, such as Napster [19], Gnutella [16], KaZaa [17], Chord [12], CAN [8], GridVine [1] and Mercury [3], to name a few. And some systems have been implemented and even widely used. However, few systems could achieve the above design goals simultaneously. To address the issues of scalability, efficiency and complex query support, in this work, we present a file sharing system, called **Peer-to-peer Semantic Overlay Network (PSON)**. Its basic design effectively exploits the concepts of hierarchy and semantics. Files shared in PSON are classified into a prior semantic hierarchy. Fig. 1 gives a simple example where each tree node represents a semantic entry. Any file in

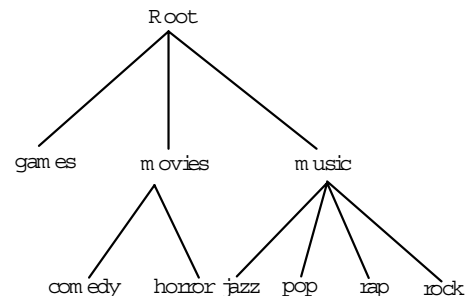


Fig. 1. An example of semantic hierarchy

the system semantically belongs to one tree node. With some attributes (such as level and label), semantic entries can be easily compared and ordered, thus we can sort all the files in the system accordingly.

Peers joining PSON are categorized into normal peers (or just peers) and super peers. Powerful nodes can serve as super peers. Normal peers connect to a super peer and together form a cluster. If we treat each cluster as a logical node, then all logical nodes form an overlay network (referred to as semantic overlay) in which queries are routed on the basis of semantics. Each of the logical nodes (i.e., clusters) has a content directory to manage. Peers belonging to a cluster share the content directory assigned to the cluster. In this way, when a peer wants to publish a file, it first extracts metadata and generates a location-metadata pair (i.e., a directory item). Then the directory item is distributed into the semantic overlay network, finds a proper “host” cluster, and finally is stored in some peer in the cluster. Similarly, when a peer wants to search some file, a search query is first issued, and then forwarded to a cluster responsible for storing the corresponding directory. By a local search in the cluster, the location for the requested item is obtained.

Clearly, in PSON, semantic overlay construction and routing is the key to the whole system design. And this is the core issue we will focus in this paper. To conserve the semantics of the shared content and facilitate overlay routing, we propose to use red-black tree as the overlay structure, for which an in-order tree walk could yield an encoded semantic tree, which can naturally help to conserve semantics. Since the searching time of red-black tree is bounded by $O(\log(n))$, the routing (logical) hops of the semantic overlay is bounded. We will describe the architecture of PSON and present our

*This work is supported in part by the National Science Foundation under Grant No. 0435230.

overlay construction and routing algorithms. We also conduct simulations, and the results show that the PSON is capable of supporting complex queries very efficiently.

The rest of the paper is organized as follows. In Section II, we review some background and related work. Then in Section III, we present the PSON architecture and discuss the basic design issues of PSON, including overlay construction, overlay routing, and system dynamics. Finally, we describe our simulation results in Section IV, and conclude the paper in Section V.

II. BACKGROUND AND RELATED WORK

In this section, we first present some background on semantic hierarchy and hierarchical P2P systems, then we describe some related work.

A. Background

1) *Semantic Hierarchy*: Content shared in a file system can usually be semantically classified into a hierarchy (referred to as “semantic hierarchy” or “classification hierarchy” in this paper) based on genres. For example, music files can be first categorized into jazz, pop, rap, and rock, etc, and rock music files can be further subdivided into “soft rock” and “hard rock”. Each subcategory can further create a sub-hierarchy based on composer and year etc. This kind of content classification is proved very useful in content organization, especially in centralized databases. However, in P2P systems, semantic hierarchy has not been widely explored yet. Crespo et al. [4] pioneered the research in this direction. In [4], nodes with semantically similar content are clustered together and form an overlay network. There is one overlay network for each semantic group in the classification hierarchy; therefore, a node joins all the overlay networks for which it has semantically related content. In this system, semantic overlay networks are not structured. Thus, *flooding or a centralized directory* is used to find the nodes that belong to those overlay networks, as results in very high query search overhead.

2) *KaZaa*: PSON explores the heterogeneity of peers, which is similar to KaZaa [17], an enormously successful P2P file sharing service. In KaZaa, peers are divided to normal peers and super peers. Super peers usually have more connection bandwidth and better availability. Normal peers connect to a super peer and together form a cluster. Each super peer keeps track of IP addresses of its descendants and the content they are sharing in an index. KaZaa is more scalable compared with Napster [19] and Gnutella [16] due to its hierarchical architecture. However, it cannot support complex queries as the queries are routed irrespective of their content. Moreover, the communication among super peers is not well organized and thus *flooding or partial pooling* is used, which is inefficient.

B. Related Work

1) *DHT Based Systems*: DHT based P2P systems (such as CAN [8], Chord [12], Pastry [9], and Tapestry [15], to name a few) provide efficient retrieval of data items (at least for

exact-match queries). The main idea behind these systems is to assign peers to hold particular content or pointers to it. A hash function is introduced to map the object being searched for to a unique identifier and the range of this function is distributed among all peers in the network. DHT based P2P systems are designed for exact queries since they do not take semantics of the query into account due to the use of hash functions.

2) *P2P Systems Supporting Semantics*: Recently, there are several proposals, such as pSearch [13], and SSW [5], which support semantics in P2P systems. In pSearch [13], semantics of a document are generated by applying Latent Semantic Indexing (LSI) on a term (or semantic) vector which is generated from the document by using Vector Space Model (VSM). CAN is then employed to create an overlay by using the document semantics as the key to store document index. In principle, this work extends the classical IR (Information Retrieval) algorithms to the P2P environment so as to provide “content” based search. In other words, the “semantics” in pSearch are basically abstracted from the document content. SSW [5] employs a similar approach to the one used in pSearch to generate semantic vectors. In SSW, peers are clustered according to the semantics of local data and self-organized as a small world overlay network. Further, a dynamic dimension reduction method is used to decrease the dimensionality of this overlay network.

The key difference between PSON and these systems is that PSON assumes a prior semantic hierarchy. To some extent, PSON is complementary to these systems. An analogy is that Yahoo classification is in fact an indispensable service to us even though we have powerful Google search. This is because that it is quite often that we do not know the right key words, but we do know what category our needs belong to.

3) *Hierarchical Directory Service Systems*: From the requirement of a prior semantic hierarchy, PSON is quite similar to some hierarchical directory service systems such as DNS [6], [7] and TerraDir [10], [11]. DNS is one of the most important services widely deployed in the Internet. It resolves domain names through lookup and search in a hierarchical distributed database. TerraDir generalized DNS and designed generic directory services for arbitrary applications. Moreover, advanced replication and caching techniques are employed to improve performance. Compared with these two systems, the key difference is on overlay construction and routing: PSON maps the semantic hierarchy to a red-black overlay tree, while TerraDir and DNS map the namespace directly to the physical network topology. The major advantage of the use of red-black tree is that the routing performance can be significantly improved, and the average search time will be strictly bounded by $O(\log(n))$.

III. PSON ARCHITECTURE

In this section, we first give an overview of PSON and discuss the principles to construct semantic overlay. Then, we present in detail the kernel design issues of PSON that includes algorithms for semantic overlay construction and query routing.

A. Overview

PSON is a self-organizing semantic overlay network, which is composed of a number of logical nodes. A logical node here stands for a cluster of peers that are geographically close to each other. A powerful peer (e.g., with more bandwidth and/or more availability) in the cluster is selected as a super peer. Different from KaZaa, the super peers are not necessary to maintain the local directories; instead, they are mainly responsible for routing in the network: communicating with other super peers and coordinating local normal peers. The overlay network among super peers is well organized based on semantics. Before demonstrating the semantic overlay network construction, the concept of semantic tree is first elaborated as follows.

1) *Semantic Tree*: Files shared in PSON are classified into a semantic hierarchy. In Fig. 1, a tree structure of the semantic hierarchy is illustrated¹. Each tree node represents a semantic entry. Each semantic entry can be denoted as a tuple with three attributes: *semantic label*, *semantic level* and *ancestor list*. For example, in Fig. 1, “Root”, “music”, “movie”, “jazz”, “rock”, etc. are all semantic labels which characterize the class of files. Semantic level is the tree level of the semantic entry in the semantic tree. Ancestor list of a semantic entry is the list of all its ancestors’ semantic labels and levels. For instance, the three attributes of “pop” would be:

Semantic Label: Pop

Semantic Level: 2

Ancestor List: Music 1, Root 0

2) *Ordering Semantic Entries*: With three attributes, semantic entries can be easily compared and ordered. For any two semantic entries, if they have the same parents, their order is decided lexicographically. If they have different parents, then find their lowest-level uncommon ancestors (according to their ancestor lists) and compare them lexicographically. For example, referring to Fig. 1, “games” is smaller than “movies” since they have the same parent “Root” and “g” is smaller than “m”. Similarly, we can say “comedy” is smaller than “pop” because their lowest-level uncommon ancestors are “movies” and “music” respectively, and “movies” is smaller than “music”.

3) *Semantic Overlay Construction*: The basic design philosophy of PSON is to construct an overlay based on semantics. In other words, PSON organizes various semantic entries into an overlay. Once we fix the overlay structure, the simplest way for overlay construction is to do a direct mapping. For example, we can easily construct an overlay tree which matches exactly the semantic tree: each super peer (or cluster) maintains the directory of files which belong to one class denoted by one semantic entry; super peers forms an overlay tree with the same shape as that of the semantic tree. However, search in such an overlay is not efficient. Moreover, we have to keep in mind that P2P file sharing systems have

¹Semantic tree is probably the most popular and useful classification hierarchy. Other classification hierarchies are possible (see [4]) of the semantic hierarchy is illustrated. In our work, we use the tree structure to demonstrate the PSON design.

high dynamics: files are inserted and deleted, and peers join and leave. To maintain a dynamic P2P system in an efficient and scalable way is very challenging. Thus, choosing a good overlay structure, and mapping the semantic tree to the overlay are the key issues in the system design.

Overlay Structure In our current design of PSON, we use balanced binary search tree (i.e., red-black tree) for overlay structure. It is possible to employ other overlay structures, such as meshes and rings. *We choose red-black tree because of its many characteristic features: in-order walk yields an ordered list; search time is bounded by $O(\log(n))$; and tree maintenance is simple.* We will investigate other overlay structures in our future work.

From Semantic Tree to Overlay Tree In PSON, the red-black overlay tree is built in an incremental fashion. In the overlay tree, each cluster (or logical node) corresponds to one semantic entry in the semantic tree. In other words, the peers in the cluster maintain the directory of the files belonging to the semantic class. However, the “mapping” clusters of semantic entries only “appear” in the overlay when necessary. For example, at the starting stage of the system, only few files are available for sharing, and there is no necessity to build a big overlay. In this case, maybe only one cluster is enough, and thus the overlay tree is actually a “degraded” tree with one node (which corresponds to the “Root” semantic entry). Later on, with more music files available, we “fork” a cluster which stores the music file directory. And the overlay now grows to a two-node tree. But how to connect the new logical node to the existing overlay? Remember we use binary search tree as overlay structure. Thus, the new node is inserted to the existing tree based on its semantic ordering. For instance, “music” is smaller than “Root”, then the node storing “music” files will be on the “left” side of the node storing “Root” files (or unclassified files). In such a manner, we can construct a semantic overlay with the form of red-black tree.

In the following, we discuss two basic PSON design issues: overlay construction and overlay routing.

B. Building a Semantic Overlay

As mentioned earlier, the overlay network is built in an incremental fashion. Clusters (or logical nodes) are inserted in such a way that they form a balanced tree structure. The insertion procedure has two parts. First, a logical node is inserted according to its semantics, i.e., based on its semantic entry; and second, an insert fixup procedure is invoked so as to ensure the red-black tree properties. The insert fixup method may include some rotations of the tree to make it balanced.

Every node in the overlay tree keeps pointers to its left child, right child, parent, smallest descendant in its left subtree, and largest descendant in its right subtree. Smallest and largest peers’ pointers are kept for query routing purposes. Algorithm 1 shows the logical node insertion procedure. In this algorithm, the pointers of a super peer “node” are denoted by “node.left”, “node.right”, “node.parent”, “node.smallest”, and “node.largest”. “root” is the root of the overlay tree, while “Root” is the root semantic entry in the semantic hierarchy.

If we go through Algorithm 1, we can conclude that in PSON overlay, semantic entry “Root” is a pre-existing node in

Algorithm 1 Overlay-Insertion(*newnode*)

```

1: // “<”, and “)” denotes semantic comparison
2: Contact bootstrap node, B
3: Add newnode’s address to the super peer list in B
4: B checks if newnode is the first super peer
5: if true then
6:   root=newnode
7:   newnode.right = Root
8:   newnode.largest = Root
9:   Root.parent = newnode
10: else
11:   Strategically select one existing super peer, and return
      address to newnode
12:   newnode contacts this super peer (referred to as
      oldnode)
13:   if newnode < oldnode then
14:     Recursively find the position of newnode in the left
      subtree of oldnode and insert it into the tree
15:   else
16:     Recursively find the position of newnode in the right
      subtree of oldnode and insert it into the tree
17:   end if
18: end if

```

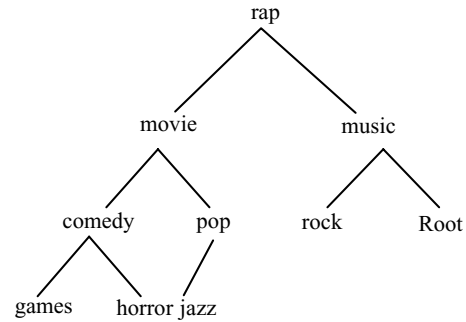


Fig. 3. The structure of the overlay tree

lies in its own left subtree by comparing “rap” semantically to its smallest (left) descendant. Since “music” does not have a left descendant in the current overlay, it checks to see if “rap” belongs to its parent’s left tree or not. “music” does not have a parent either because it itself is the root of the tree; thus, it makes “rap” its own left child. “movie” and “pop” are then inserted by their respective semantic parents, “Root” and “music” in similar procedures. Clearly, the overlay tree is not balanced after the insertions and we now need to make it balanced so as to be able to support efficient routing.

A right rotation is performed by “rap” to restore the balance of the overlay tree. The subtree at “rap” gets one level closer to root and the subtree at “music” gets one level further from the root of the overlay tree. As shown in Fig. 2, the final overlay tree is balanced. In our algorithms, both right and left rotations are local operations, and they run in $O(1)$ time. Rotations are called when nodes are inserted/deleted and the tree becomes unbalanced. Since each node in a red-black tree maintains a “color” bit, no global information (for example, the height of the (sub)tree) is needed to decide if a rotation should be performed or not. Thus, rotations in PSON overlay trees are distributed.

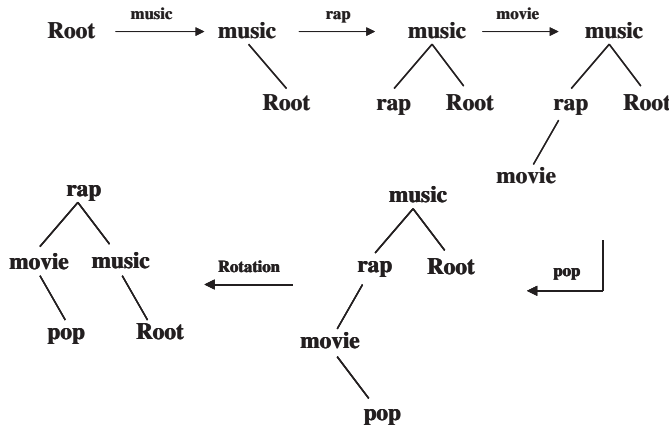


Fig. 2. An example of overlay node insertion

the system with level 0, and a new node can only be created by its semantic parent. This design is motivated by our intuition: if there are few files in one category, then it is not necessary to construct a new overlay node to store the corresponding directory information and the parent category node should be sufficient. Now we explain the overlay construction algorithm with the help of an example (depicted in Fig. 2). Initially, “Root” is the only node in the system and handles all kinds of files. Then suppose that the number of “music” files increases beyond a certain threshold. “Root” initiates a new node insertion in the overlay that has semantic label “music” and level 1. The directory of music files is given to this new node which can further create its own semantic children. “rap” is the next node to be inserted and its insertion is initiated by its semantic parent “music”. “music” finds the position of “rap” as follows: “music” first checks to see whether “rap”

Maintaining the Semantics In Fig. 2, the overlay nodes are inserted in the following order : “Root”, “music”, “rap”, “movie”, “pop”. Fig. 3 is resulted when all nodes of Fig. 1 are inserted. Its in-order walk is: “games”, “comedy”, “horror”, “movie”, “jazz”, “pop”, “rap”, “rock”, “music”, “Root”. Though the overlay tree depends on the order in which individual nodes are inserted, it is guaranteed that for the same nodes, in-order traversal of tree yields the same semantic ordering of the semantic entries irrespective of the insertion order, thus keeping the semantics.

Overlay Node Deletion The overhead of overlay node deletion is comparatively less than that of insertion. The reason is that for insertion, the new node’s position in the overlay is searched first and then the fixup is performed. While in the deletion case, the node leaves the overlay directly (at the maximum, it tells its semantic parent that it is leaving the overlay). Then, a deletion fixup mechanism is invoked to restore the red-black tree properties of the overlay which may again involve some rotations.

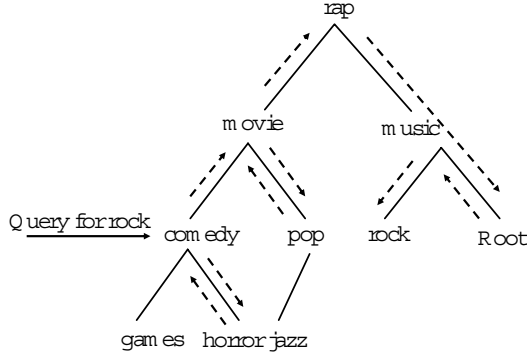


Fig. 4. An example of query routing. Dashed arrows denote the query path.

C. Overlay Query Routing

When a normal peer issues a search query to the system, if its local cluster does not have the requested files, it will contact its super peer, which in turn initiates an overlay query routing procedure. Our overlay query routing algorithm is shown in Algorithm 2.

Algorithm 2 `Overlay_Query(current_node, query_node)`

```

1: // ⟨,⟩ denotes semantic comparison
2: if current_node ⟨ query_node then
3:   if query_node ⟨ current_node.smallest ||
     current_node.smallest == null then
4:     pass the query to current_node.parent
5:   else
6:     pass the query to current_node.left
7:   end if
8: else
9:   if query_node ⟨ current_node.largest ||
     current_node.largest == null then
10:    pass the query to current_node.parent
11:   else
12:    pass the query to current_node.right
13:   end if
14: else
15:   the query is resolved
16: end if

```

An example is illustrated in Fig. 4. Consider a query for “rock” songs from a cluster that is responsible for the directory of “comedy” movies. “comedy” is compared with “rock” (Line 2) and since “rock” is greater than “comedy”, it is now compared with the largest descendant of its right subtree at “comedy” (Line 9) i.e. “horror”. “rock” is even greater than “horror”, therefore, it is further forwarded to the parent of “comedy”, “movie” (Line 10). “movie” performs the same comparison again (Line 2) and routes the query accordingly.

D. Complexity Analysis

From the above description, we can easily know that the complexity of both the overlay construction and routing algorithms is $O(\log(n))$ due to the fact that red-black tree

is utilized as the overlay structure (where n is the number of overlay nodes). If we assume there are totally N peers, and the average cluster size is M , then the average search path length is $O(\log(N/M))$. If we compare with the average search path length of SSW², which is $O(\frac{\log^2(2N/M)}{l})$ (where l is a constant, and is set as 4 in the evaluation of SSW [5]), we can conclude that query search in PSON is much faster than SSW in average.

E. Other Design Issues

Here we identify several other important issues to be considered in the PSON design.

- **System Dynamics:** One of the key characteristics of a P2P system is high dynamics: peers come and go, and files are inserted and deleted. In PSON, how to handle peer joining and leaving? How to deal with file insertion and deletion? How to control the number of peers in each cluster? How to balance files in different clusters? And further, how will these system dynamics affect the overlay performance?
- **Fault Tolerance:** In P2P systems, it is quite often that nodes malfunction or stop working. When a peer fails or even super peer fails, the PSON overlay should still function without much performance penalty. To ensure the robustness of PSON, what additional measures should be taken?
- **Load Balancing:** Since PSON’s overlay is structured as a balanced binary search tree, naturally half of the queries will traverse from one half of the tree to other half. This may lead to a lot of query traffic at root and at high level nodes of the overlay tree. How to well balance traffic while maintaining a well-structured overlay?

Though these issues are indispensable to the integrated design of PSON, they are out of the scope of this paper. We have presented some basic ideas in our technical report [2]. More in-depth studies are underway.

IV. SIMULATION STUDY

In this section, we evaluate the performance of PSON using simulations.

A. Simulation Setup

We implement PSON in NS2 [18], and conduct the simulation experiments on Redhat Linux platform. The physical network topologies are generated using GT-ITM[14].

The simulation is initialized by having one node (with semantic label as “Root”) pre-exist in the network acting as the root of the insertion tree and then injecting node join operations into the network till the network reaches a certain size. Afterwards, query search events are injected into the network. At this point only, we collect the following metrics:

- **Search Delay** is the delay in searching a requested (class of) file(s). We measure this delay by averaging the

²We have identified the fundamental difference between SSW and PSON in Section II. Here, we simply compare the average search path length of the two systems.

number of logical hops traversed by a query from the source to the destination.

- **Search Success Ratio** is the percentage of successful searches that are able to locate existing files in the overlay network.

Now we describe some important issues related to our simulation experiments: semantic tree generation, query generation, and query classification.

1) *Semantic Tree Generation*: In the simulations, we generate random words of 5 letters each. A random word stands for the class of files (such as “Music”, “Comedy”) a logical node would be responsible for, and it characterizes the label of a semantic tree node. For the first node, a random number (within a range) of random words are generated as the semantic children. Then for each child, a similar procedure is repeated until the desired number of semantic tree nodes are generated. We also control the height of the semantic tree. In our simulations, we set the number of semantic tree nodes as 1000, and the maximum height of the tree as 20. In addition, the number of children for a tree node can also be controlled. We set this maximum number as 5 unless clarified otherwise.

2) *Query Generation*: To closely mimic query behaviors in real world, we generate overlay queries in the following way: We let every node in the system maintain a file containing all of the semantic entries. Whenever a node is prompted to issue a query, it randomly reads a semantic entry from the file, and then releases the selected semantic entry as the query into the overlay network.

3) *Query Classification*: We design two types of queries: exact queries and complex queries. An exact query with semantic entry α requires the system to return the files (more precisely, the addresses of the peers which contain the files) which exactly match α . While for a complex query with semantic entry β , the system should return the files which match β as well as the files match any of the “descendant” semantic entries of β . For instance, if “music” and two of its semantic children, “jazz” and “classical” are present in the overlay then a complex query for “music” will return “music”, “jazz” and “classical”. This is because a complex query for “music” most probably means that *I do not know what music files to download, but I want to browse all music files*. Thus “jazz” and “classical” which are semantic descendants of “music” should also be returned.

Since overlay queries correspond to the nodes of the semantic hierarchy, an exact query is termed as successful if the corresponding node is found and unsuccessful if it is not found in the overlay. For a complex query, it is termed as successful if it finds all of the existing nodes classified as the query node. For instance, “music”, “pop” and “rap” are the only music nodes in the overlay. When a query for “music” returns these three, we will say that the query is 100% successful as it finds all the nodes that can be classified as “music”.

It should be noted that PSON can indirectly handle arbitrary queries, such as “join”, “union”, “select”, and “projection”. These types of queries should be first decomposed into multiple queries (including exact and complex queries) which can be routed directly in PSON. In other words, a relatively powerful user interface at each peer is required to be able to

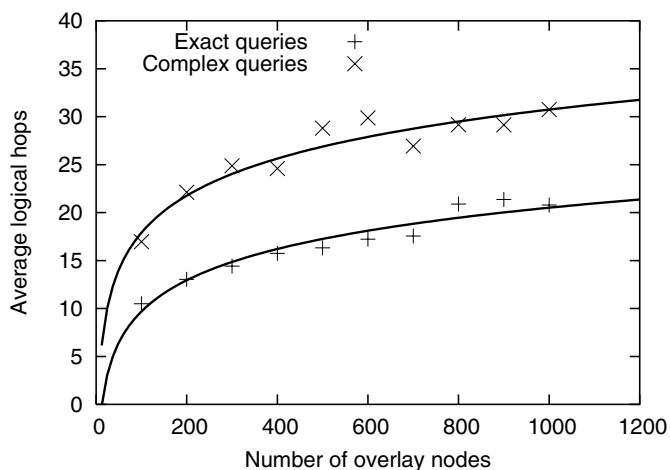


Fig. 5. Search delay with varying overlay size

decompose arbitrary queries, collect search results and conduct post-query processing. From the realization point of view, such a user interface is not difficult to implement. On the other hand, how to incorporate in-network query processing in PSON is an interesting topic in our future research plan.

B. Query Performance

Exact Query Performance First we present the performance of exact match queries in PSON. It is worth mentioning here that the search success ratio is 1 for every overlay query in PSON. If the logical node being searched is present, then the PSON search mechanism guarantees that it will be found in $O(\log n)$ hops. The lower curve in Fig. 5 shows the average number of logical hops traversed (i.e., the search delay) for exact queries. We can observe that the query performance is nicely bounded and hence scalable, which is consistent with our early analysis that the overlay tree is always balanced.

Complex Query Performance Now we show the performance of complex queries in PSON.

The complex query performance curve is also plotted in Fig. 5. Compared with exact queries, complex queries require a larger number of logical hops on average. This is consistent with our assertion that for a complex query, we need to traverse more nodes as the query does not supply enough attributes. However, the shape of the two curves are similar (both are logarithmic), which indicates that the lookup time of complex queries is again bounded. Thus, PSON is able to support complex queries efficiently while maintaining scalability and search efficiency.

C. Comparison of PSON and Flooding Overlay

In this experiment, we compare PSON with flooding overlay (used in SON [4], which is the closest work to ours in semantic overlay P2P system design) for both exact and complex queries.

In flooding overlay, a query is forwarded to a node’s all known neighbors. The neighbors check to see whether they can

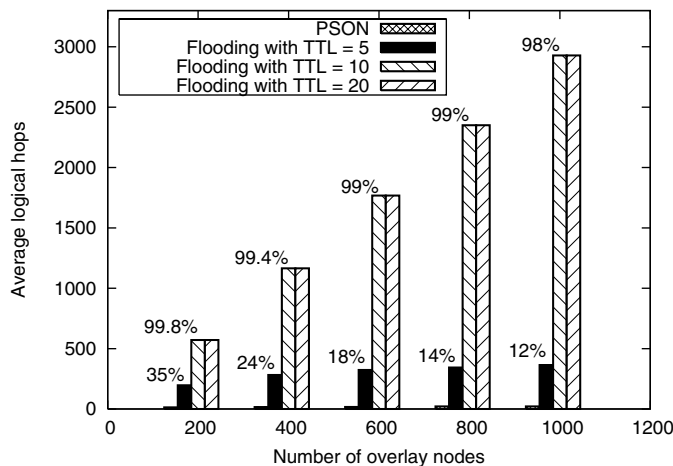


Fig. 6. Comparing exact query performance of PSON and flooding overlay

reply to the query. If they find a match, they reply; otherwise they forward query to their neighbors and so on. In this way, for each query the whole overlay network is searched, which is clearly not scalable. One way to curtail the search process is to use time-to-live (TTL) controlled flooding mechanisms. Whenever a node forwards a query, it increments the hop count of the query. If the hop count exceeds the TTL field of the message, the node stops forwarding the query. This TTL value controls the query propagation in the system, but it may fail to find a query even if it exists in the system. Hence, the TTL field decreases the successful percentage of queries.

In the exact queries case, flooding overlay is evaluated against three different values of TTL, 5, 10 and 20. Fig. 6 shows the search time in flooding overlay as well as in PSON for varying number of overlay nodes. In flooding overlay, the accuracy of queries decreases with the number of overlay nodes for the same TTL value (except for TTL = 20). This is because the percentage of nodes contacted by a query message decreases with increasing overlay size. PSON, on the other hand, achieves 100% successful search for all sizes of overlay. In addition, even though the successful percentage of flooding overlay with TTL 20 is also 100%, PSON achieves the same performance with even less than five percent of logical hops needed by flooding overlay.

For the complex query case, we obtain similar results, which are omitted from this paper due to space limit.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a powerful P2P file sharing system, PSON. It satisfies all the three design goals of a successful P2P file sharing system.

- Scalability: PSON is not only scalable to a large number of peers, it is also scalable to a large amount of traffic.
- Routing Efficiency: By utilizing a red-black overlay tree, PSON could route queries in time bounded by $O(\log(n))$.
- Supporting Complex Queries: By exploring the concept of semantics, PSON could support various complex queries very effectively.

We have described the PSON architecture, and addressed the key design issues: overlay construction and overlay routing. By simulations, we have shown PSON is very efficient in both exact query and complex query routing.

To make PSON a real system, there are still many issues to explore. Besides the issues of system dynamics, fault tolerance, and load balancing mentioned in Section III-E, there are several other directions worth future investigation: 1) Implementing a full-fledged PSON system would help to further evaluate the system performance and facilitate the system deployment; 2) PSON does not consider geography-aware overlay routing, as obviously might lead to long search delay even though the number of logical hops is bounded. Designing an overlay routing algorithm which could incorporate the physical topology is definitely desirable.

REFERENCES

- [1] K. Aberer, P. Cudré-Mauroux, M. Hauswirth, and T. V. Pelt, "Gridvine: Building internet-scale semantic overlay networks." in *International Semantic Web Conference*, 2004, pp. 107–121.
- [2] J. Ahuja, J.-H. Cui, S. Chen, and L. Lao, "A scalable peer-to-peer file sharing system supporting complex queries," *UCONN CSE Technical Report: UbiNet TR05-01 (BECAT/CSE-TR-05-4)*, p. null, January 2005.
- [3] A. R. Bhamambe, M. Agrawal, and S. Seshan, "Mercury: supporting scalable multi-attribute range queries," in *SIGCOMM*, 2004.
- [4] A. Crespo and H. Garcia-Molina, "Semantic overlay networks for p2p systems," in *International Workshop on Agents and Peer-to-Peer Computing (AP2PC'04)*, 2004, pp. 1–13.
- [5] M. Li, W.-C. Lee, and A. Sivasubramanian, "Semantic small world: An overlay network for peer-to-peer search." in *ICNP*, 2004, pp. 228–238.
- [6] P. V. Mockapetris, "Domain names - concepts and facilities," request for Comments 1034, Internet Engineering Task Force, November 1987.
- [7] —, "Domain names - implementation and specification," request for Comments 1035, Internet Engineering Task Force, November 1987.
- [8] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in *SIGCOMM*. ACM Press, 2001, pp. 161–172.
- [9] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," *Lecture Notes in Computer Science*, vol. 2218, 2001.
- [10] B. Silaghi, B. Bhattacharjee, and P. Keleher, "Query routing in the terradir distributed directory," in *SPIE ITCOM*, August 2002.
- [11] B. Silaghi, V. Gopalakrishnan, B. Bhattacharjee, and P. Keleher, "Hierarchical routing with soft-state replicas in terradir," in *The 18th International Parallel and Distributed Processing Symposium*, April 2004.
- [12] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *ACM SIGCOMM*, August 2001, pp. 149–160.
- [13] C. Tang, Z. Xu, and S. Dwarkadas, "Peer-to-peer information retrieval using self-organizing semantic overlay networks," in *SIGCOMM*. ACM Press, 2003.
- [14] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *IEEE Infocom*, vol. 2, March 1996, pp. 594–602.
- [15] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," UC Berkeley, Tech. Rep. UCB/CSD-01-1141, apr 2001.
- [16] Gnutella, <http://gnutella.wego.com/>.
- [17] KaZaA, <http://kazaa.com/>.
- [18] NS2, <http://www.isi.edu/nsnam/ns/>.
- [19] Napster, <http://napster.com/>.