Semantic Overlay Networks for P2P Systems

Arturo Crespo and Hector Garcia-Molina

Google Technologies Inc., Stanford University
crespo@google.com, hector@cs.stanford.edu

Abstract. In a peer-to-peer (P2P) system, nodes typically connect to a small set of random nodes (their neighbors), and queries are propagated along these connections. Such query flooding tends to be very expensive. We propose that node connections be influenced by content, so that for example, nodes having many "Jazz" files will connect to other similar nodes. Thus, semantically related nodes form a Semantic Overlay Network (SON). Queries are routed to the appropriate SONs, increasing the chances that matching files will be found quickly, and reducing the search load on nodes that have unrelated content. We have evaluated SONs by using an actual snapshot of music-sharing clients. Our results show that SONs can significantly improve query performance while at the same time allowing users to decide what content to put in their computers and to whom to connect.

1 Introduction

Peer-to-peer systems (P2P) have grown dramatically in recent years. They offer the potential for low cost sharing of information, autonomy, and privacy. However, query processing in current P2P systems is very inefficient and does not scale well. The inefficiency arises because most P2P systems create a random overlay network where queries are blindly forwarded from node to node. As an alternative, there have been proposals for "rigid" P2P systems that place content at nodes based on hash functions, thus making it easier to locate content later on (e.g., [14, 8]). Although such schemes provide good performance for point queries (where the search key is known exactly), they are not as effective for approximate, range, or text queries. Furthermore, in general, nodes may not be willing to accept arbitrary content nor arbitrary connections from others.

In this paper we propose Semantic Overlay Networks (SONs), a flexible network organization that improves query performance while maintaining a high degree of node autonomy. With Semantic Overlay Networks (SONs), nodes with semantically similar content are "clustered" together. To illustrate, consider Figure 1 which shows eight nodes, A to H, connected by the solid lines. When using SONs, nodes connect to other nodes that have semantically similar content. For example, nodes A, B, and C all have "Rock" songs, so they establish connections among them. Similarly, nodes C, E, and F have "Rap" songs, so they cluster close to each other. Note that we do not mandate how connections are done inside a SON. For instance, in the Rap SON node C is not required to connect directly to F. Furthermore, nodes can belong to more than one SON (e.g., C belongs to the Rap and Rock SONs). In addition to the simple partitioning illustrated

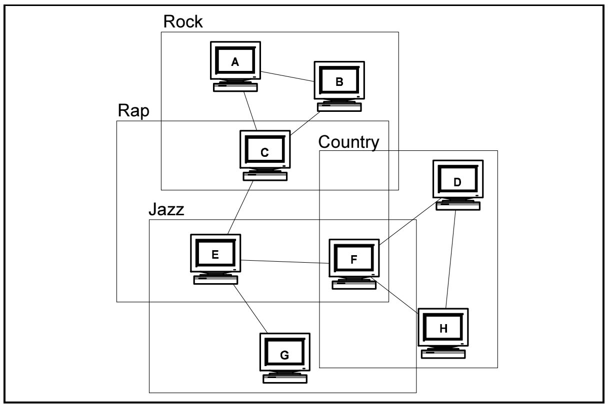


Fig. 1. Semantic Overlay Networks

by Figure 1, in this paper we will also explore the use of content hierarchies, where for example, the Rock SON is subdivided into “Soft Rock” and “Hard Rock.”

In a SON system, queries are processed by identifying which SON (or SONs) are better suited to answer it. Then the query is sent to a node in those SONs and the query is forwarded only to the other members of that SON. In this way, a query for Rock songs will go directly to the nodes that have Rock content (which are likely to have answers for it), reducing the time that it takes to answer the query. Almost as important, nodes outside the Rock SON (and therefore unlikely to have answers) are not bothered with that query, freeing resources that can be used to improve the performance of other queries.

There are many challenges when building SONs. First, we need to be able to classify queries and nodes (what does “contain rock songs” means?). We need to decide the level of granularity for the classification (e.g., just rock songs versus soft, pop, and metal rock) as too little granularity will not generate enough locality, while too much would increase maintenance costs. We need to decide when a node should join a SON (if a node has just a couple of documents on “rock,” do we need to place it in the same SON as a node that has hundreds of “rock” documents?). Finally, we need to choose which SONs to use when answering a query.

Many of our questions can only be answered empirically by studying real P2P content and how well it can be organized into SONs. For our empirical evaluation we have chosen music-sharing systems. These systems are of interest not only because they are the biggest P2P application ever deployed, but also because music semantics are rich enough to allow different classification hierarchies. In addition there is a significant amount of data available that allows us to perform realistic evaluations. While our experimental results in this paper are particular to this important application, we have no reason to believe they would not apply in other applications with good classification hierarchies.

Also note that due to space limitations, in this paper we do not present the full results of our work. A more detailed and formal description of our approach, as well as additional experimental results, can be found in the extended version of this paper [1].

2 Related Work

The idea of placing data in nodes close to where relevant queries originate was used in early distributed database systems [3]. However, the algorithms used for distributed databases are based on two fundamental assumptions that are not applicable to P2P systems: that there are a small number of stables nodes, and that the designer has total control over the data. There are a number of P2P research systems (CAN [8], CHORD [14], Oceanstore [4], Pastry [10], and Tapestry [21]) that are designed so documents can be found with a very small number of messages. However, all these techniques either mandate a specific network structure or assume total control over the location of the data. Although these techniques may be appropriate in some application, the lack of node autonomy has prevented their use in wide-scale P2P systems. There is a large corpus of work on document clustering using hierarchical systems (see [5] for a survey). However, most clustering algorithms assume that documents are part of a controlled collection located at a central database. Clustering algorithms for decentralized environments have also been studied in the context of the web. However, these techniques depend on crawling the data into a centralized site and then using clustering techniques to either make web search results more accurate (as in SONIA [11]) or easier to understand (as in Vivisimo [17]). A more decentralized approach has been taken by Edutella [6] and HyperCup [13] where peers with similar content connect to the same super peer.

3 Semantic Overlay Networks

In this section we introduce informally the concept of Semantic Overlay Networks (SONs) (see [1] for a formal definition). In a P2P system, the links between the nodes typically form a single overlay network. In this paper we advocate the creation of multiple overlay networks to improve search performance. We do *not* focus on on how queries are routed within an overlay network (see Section 2 for a brief overview of current solutions to the intra-overlay network routing problem). Therefore, we will ignore the link structure within an overlay network and we will represent an overlay network just by the set of nodes in it.

Requests for documents are made by issuing a query q and some additional system-dependent information (such as the horizon of the query). A query is also system dependent and it can be as simple as a document identifier, or keywords, or even a complex SQL query. In this paper we assume that queries are partial, so the request includes a minimum number of results that need to be returned.

3.1 Classification Hierarchies

Our objective is to define a set of overlay networks in such a way that, when given a request, we can select a small number of overlay networks whose nodes have a "high" number of hits. The benefit of this strategy is two fold. First, the nodes to which the request is sent will have many matches, so the request is answered faster; and second, but not less important, the nodes that have few results for this query will not receive it, avoiding wasting resources on that request (and allowing other requests to be processed faster).

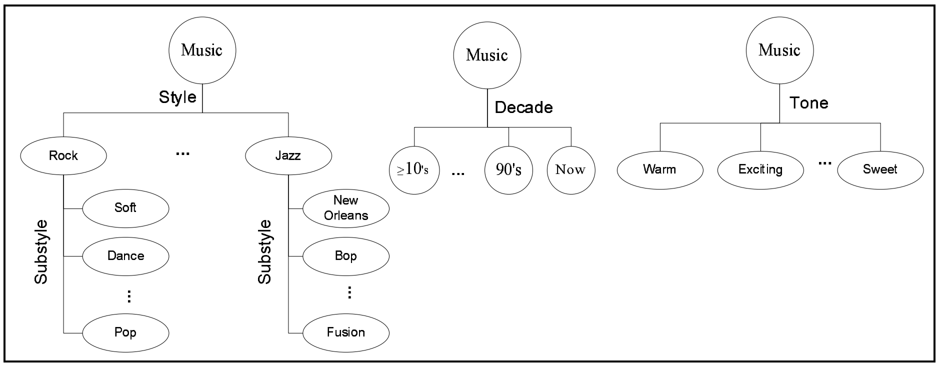


Fig. 2. Classification Hierarchies

We propose using a classification hierarchy as the basis of the formation of the overlay networks. For example, in Figure 2, we show 3 possible classification hierarchies for music documents. In the first one, music documents are classified according to their style (rock, jazz, etc.) and their substyle (soft, dance, etc.); in the second one, they are classified by decade; and in the third one, they are classified by tone (warm, exciting, etc.).

Each document and query is classified into one or more *leaf* concepts in the hierarchy. However, in practice, classification procedures may be *imprecise* as they may not able to determine exactly to which concept a query or document belongs. In this case, imprecise classification functions may return non-leaf concepts, meaning that document or query belongs to one or more descendant of the non-leaf concept, but the classifier cannot determine which one. For example, when using the leftmost classification hierarchy of Figure 2, a "Pop" document may be classified as "Rock" if the classifier cannot determine to which substyle ("Pop," "Dance," or "Soft") the document actually belongs. Classifiers may also make *mistakes* by returning the wrong concept for a query or document. We call the set of documents that classify into the same concept the "bucket" of that concept.

In a P2P system, documents are actually kept by nodes. Therefore, we need to classify nodes, rather than documents. We call a group of semantically related nodes a Semantic Overlay Network (SON), and we associate each SON with a concept in the classification hierarchy. We call a SON associated with concept c, the SON for c or SON_c. For example, in the leftmost hierarchy in Figure 2 (if we assume that only the its only concepts are the ones shown), we will define at 9 SONs: 6 associated with the leaf nodes (soft, dance, pop, New Orleans, etc.), one associated with rock, another associate with jazz, and a final one associate with music. To completely define a SON, we need to explain how nodes are assigned to SONs and how we decide which SONs to use to answer a query.

A node decides which SONs to join based on the classification of its documents. There are many strategies for node placement. For example, we may place a node in SON_c if it has any document classified in c. This strategy is very conservative as it will place a node in SON_c if just one document classifies as c. A less conservative strategy will place a node in SON_c if a "significant" number of document classifies as c. Such

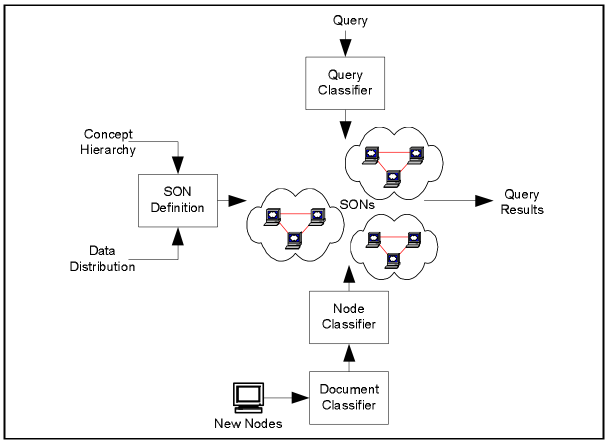


Fig. 3. Generating Semantic Overlay Networks

less-conservative strategy has two effects: it reduces the number of nodes in a SON and it reduces the number of SONs to which a node belongs. The first of these effects increases the advantages of SONs as fewer nodes need to be queried. The second effect reduces the cost of SONs as the greater the number of SONs to which a node belongs, the greater the the node overhead for handling many different connections. However, a less conservative strategy may prevent us from finding all documents that match a query. In Section 6, we study different strategies for assignment of nodes to SONs.

After assigning nodes to SON, we may make adjustments to the SONs based on the actual data distributions in the nodes. For example, if we observe that a SON contains only a very small number of nodes, we may want to consolidate that SON with a sibling or its parent in order to reduce overhead.

To summarize, the process of building and using SONs is depicted in Figure 3. First, we evaluate potential classification hierarchies using the actual data distributions in the nodes (or a sample of them) and find a good hierarchy. This hierarchy will be stored by all (or some) of the nodes in the system and it is used to define the SONs. A node joining the system, first floods the network with requests for the hierarchy in a Gnutella fashion (we do not address security problems in this paper, but inconsistent hierarchies may be detected by obtaining the hierarchy from multiple sources and using a majority rule). Then, the node runs a document classifier based on the hierarchy obtained on all its documents. Then, a node classifier assigns the node to specific SONs (by, for example, using the conservative strategy described in this section). The node joins each SON by finding nodes that belong to those SONs. This can be done again in a Gnutella fashion (flooding the network until nodes in that SON are found) or by using a central directory. When the node issues a query, first it classifies it and sends it to the appropriate SONs (nodes in those SONs can be found in a similar fashion as when the node connected to its SON). After the query is sent to the appropriate SONs, nodes within the SON find matches by using some propagation mechanism (such as Gnutella flooding or super peers).

In the next sections, we will study the challenges and present solutions for building a P2P system using Semantic Overlay networks. We will evaluate our solutions by simulating a music-sharing system based on real data from Napster [19] and OpenNap [16]. Specifically, in this paper we will address the following challenges:

- Classification hierarchies for SONs (Section 4): If nodes have very diverse files, there will not be enough clustering to merit the use of SONs. So, in practice, will we see enough clustering? What hierarchies will yield the most clustering and the best SON organization?
- Classifying queries and documents (Section 5): Imprecise classifiers can map too many documents and queries to higher levels of the hierarchy, making searches more expensive. What are the options for building classifiers? Are they precise enough for our needs? What is the impact of classification errors?
- SON membership (Section 6): When should a node join a SON? What is the cost of joining a SON? Can we reduce the number of SONs that a node needs to belong to (while being able to find most results)?
- Searching SONs (Section 7): How do we search SONs? Is it worth having Semantic Overlay Networks? Is the search performance of a SON-based system better than a single-overlay network system such as Gnutella?

4 Classification Hierarchies

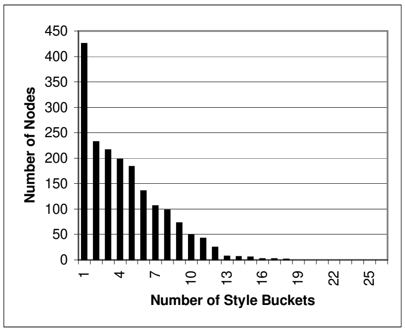
In this section we present the challenges and some solutions to the problem of choosing a good classification hierarchy for a SON-based system. Specifically, we will define what a good classification hierarchy is, how can we evaluate a classification hierarchy, and how can we choose among a set of possible hierarchies.

A good classification hierarchy is one that: (i) produces buckets with documents that belong to a small number of nodes, (ii) nodes have documents in a small number of buckets, and (iii) it allows for easy-to-implement classification algorithms that make a low number of errors (or no errors at all). (See [1] for the rationale behind these criteria.) Using this criteria, we can evaluate classification hierarchies (with the final objective of choosing the best one). This evaluation is a very important step as we have seen that if we are not careful in choosing a good classification hierarchy we may reduce or even eliminate the benefits of using SONs. To evaluate, first, we need to make sure that classifiers can be implemented and that they are efficient. Then, we use the actual data from the nodes in the system to predict the size of the SONs as well as the number of SONs to which a node will belong.

4.1 Experiments

To illustrate the issues described in this section, we evaluate the three music classification hierarchies illustrated earlier in Figure 2. In that figure we only present a small subset of the concepts in each classification hierarchy. The full sets of concepts are presented in the extended version of this paper [1] and are based on the hierarchy used by *All Music Guide* [18], a music database maintained by volunteers who manually classify songs and artist.

The first classification hierarchy divides music files according first to their style (e.g., Rock, Jazz, Classic, etc.), and then to their substyle (e.g., Soft Rock, Dance Rock,



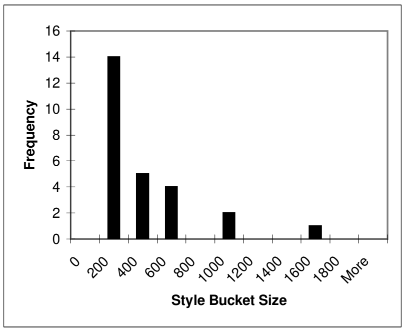


Fig. 4. Distribution of Style Buckets

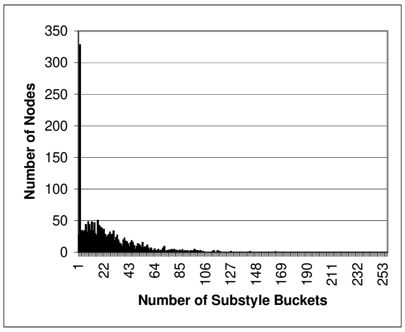
Fig. 5. Bucket Size Distribution for Style Hierarchy

etc.). For style, there are a total of 26 categories and a music file can only belong to one category; while for substyle, there are 255 categories and a file can be classified in multiple substyles. The second hierarchy classifies music files based on the decade on which the piece was originally published (10's or before, 20's, ..., 80's, and 90's or newer). Music files can only be classified in one decade. Finally, the third classification hierarchy divides files according to the "tone" of the piece (e.g., warm, exciting, sweet, energetic, party, etc.). There are a total of 128 tones and a music file can be classified in multiple tones.

In our experiment, we used the crawl of 1800 Napster nodes made at the University of Washington during the month of May 2001 [12]. This crawl included the identity of the node (user name), and for each node, the listing of its files. For most nodes, filenames were of the form "directory/author-song title.mp3" which allowed us to easily classify files by author and song titles. There was additional information (length of file, bit rate, and a signature of the content) that was not used in our evaluations. Actual file content was not available.

To evaluate the style/substyle classification hierarchy, we will first evaluate the style classification hierarchy by itself and then (if needed) we will add to the evaluation the substyle dimension. In Figure 4, we show the distribution of Style buckets. To generate this graph, for each node we counted the number of style categories for which the node had one or more files. Then we counted the number of nodes with the same number of style categories and plotted it on the graph. For example, if a node had files in the Rock, Jazz, Country, and Classic styles (and no files in the other styles), then the node would have be counted in the bar for "4 style" buckets. ¿From the graph, we can see that 425 nodes (about 24% of the total nodes) have files in just one style. Moreover, 90% of the nodes have files in eight or fewer style categories. This result means that if we define a SON based on the style of files, most nodes will have to handle very few connections.

As indicated before, the smaller the SON, the better query performance will be. However, we cannot compute the size of the Style SONs without the specific node-to-SON assignment strategy. Therefore, we will assume the most conservative strategy: a node will belong to a Style SON if it has one or more files in that Style bucket. Figure 5 shows a histogram for the number of nodes that have one or more files in each Style bucket. To generate this graph we counted, for each style, the number of nodes that have



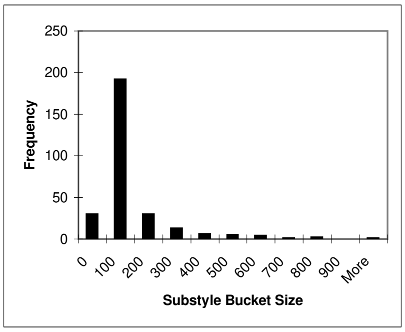


Fig. 6. Distribution of Substyle Buckets

Fig. 7. Bucket Size Distribution for Substyle Hierarchy

one or more files classified in that style. We then counted how many styles had a number of nodes in the ranges 0 to 199, 200 to 399, and so on, and plotted them on the graph. For example, the leftmost bar in the graph means that 14 styles buckets had documents that belonged to between 200 and 399 nodes. The high frequency for bucket size in the interval [200,399] is good news as it shows that the maximum size of most SONs will be small with only 11% to 22% of the nodes. However, there is one style bucket (shown by the rightmost bar) that has documents belonging to between 1600 and 1800 nodes. Thus, almost all nodes in the system have one or more documents for that bucket (this bucket corresponds to the style “Rock”). Given that there is little advantage on creating a SON based on the style “Rock,” we need to explore if it is possible to subdivide it further by using substyles.

We now consider SONs based on the substyle classification. Although the previous analysis pointed that we only needed to subdivide the Rock style category (and perhaps the 2 other categories with documents belonging to between 1000 and 1200 nodes), for completeness we will analyze all substyles categories. In Figure 6 we now show the substyle distribution, analogous to Figure 4. ¿From the graph, we can see that 328 nodes (about 18% of the total nodes) have files in just one substyle. Moreover, 90% of the nodes have files in 30 or less substyle categories. These results are again positive as it shows that number of SONs to which most nodes may belong is small. In Figure 7 we show the bucket size histogram, analogous to Figure 5. ¿From the figure we can see that 222 of the substyles (87% of the total) will have documents belonging to less than 400 nodes. However, there are again a few substyle categories that will have documents that belong to a large number of nodes, but this problem is not as bad as the one that we had when using the style classification hierarchy by itself. In particular, the category with the most number of nodes, “Alternative Pop Rock,” (which is represented by the rightmost bar in the histogram) will have documents belonging to only 1031 nodes (57% nodes). Even though the “Alternative Pop Rock” SON will have many nodes, it is still half the size of a full Gnutella network that links all the nodes. In conclusion, a combined style and substyle classification hierarchy is a good candidate for defining SONs as the maximum number of SONs that a node needs to join is small and the maximum number of nodes in a SON is also relatively small.

We also analyzed the usa of Decades as a criteria for classifying documents (graph not shown). Although most nodes had documents in only a few decade buckets, we found that more than half of the SONs will have more than 600 nodes. In fact, almost all nodes will have documents for the 70s, 80s, and 90s buckets. Therefore, given that we do not have a way of subdividing those decades, we have to reject the decade classification hierarchy.

When analyzing the the distribution of tone buckets (graph not shown), we found that the median number of buckets for which a node has documents is 43, which will result in nodes belonging to a high number of SONs. However, we also found that most buckets will contain documents belonging to a relatively small number of nodes. Specifically, 60% of the buckets will have documents belonging to 625 or fewer nodes, and 90% of the buckets will have documents belonging to 875 nodes. In conclusion, using a classification hierarchy based in tone is borderline and depending on the specifics of the tradeoff between nodes maintaining a large number of connections and the benefits of relatively small SONs, we may decide to use it or not. Nevertheless, of all the classification hierarchies evaluated, the one based on style/substyle is clearly superior and we will use it in the rest of our experiments.

5 Classifying Queries and Documents

In this section we describe how documents and queries are classified. Although the problem of classifying documents and the problem of classifying queries are very similar, the *requirements* for the document and query classifiers can be very different. Specifically, it is reasonable to expect that nodes will join a relatively stable P2P network at a low rate (a few per minute); while we could expect a much higher query rate (hundreds or even more per second). Additionally, node classification is more bursty as when a node joins the network it may have hundreds of documents to be classified; on the other hand, queries will likely to arrive at a more regular rate. Under these conditions, the document classifier can use a very precise (but time consuming) algorithm that can process in batch a large number of documents; while, the query classifier must be implemented by a fast algorithm that may have to be imprecise.

The classification of documents and queries can be done automatically, manually, or by a hybrid processes. Examples of automatic classifiers include text matching [7], Bayesian networks [9], and clustering algorithms [15]. These automatic techniques have been extensively studied and they are beyond the scope of this paper. Manual classification may be achieved by requiring users to tag each query with the style or substyle of the intended results. If the user does not know the substyle or style of the potential results, he can always select the root of the hierarchy so all nodes are queried. Finally, hybrid classifiers aid the manual classification with databases as we will see shortly in our experiments.

5.1 Evaluating our Document Classifier

Documents were classified by probing the database of All Music Guide at allmusic.com [18]. In this database songs and artists are classified using a hierarchy of style/substyle concepts equivalent to the leftmost classification hierarchy of Figure 2. Recall that for

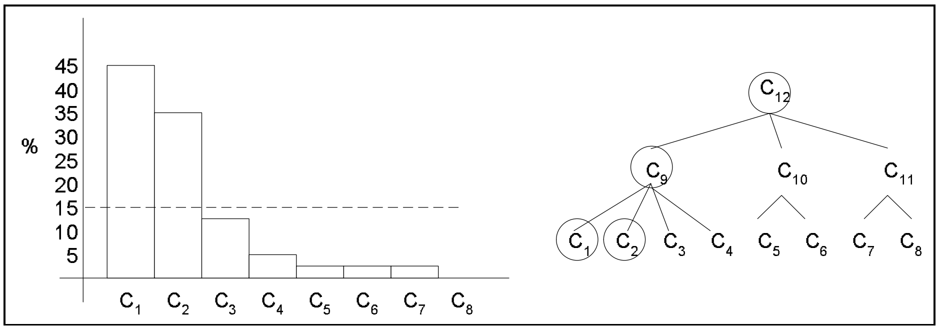


Fig. 8. Choosing SONs to join

each Napster node used in our evaluation we had a list of filenames with the format "directory/author-song title.mp3." As a first step, the document classifier extracted the author and the song title for the file. The classifier then probed the database with that author and song and obtained a list of possible song matches. Finally, the classifier selected the highest rank song and found its style and substyles. If there were not matches in the database, the classifier assigned "unknown" to the style and substyle of the file.

There were many sources of errors when using our document classifier. First, the format of the files may not follow the expected standard, so the extraction of the author and song title may return erroneous values. Second, we assumed that all files were music (but Napster could be, and was actually used, to share other kind of files). Third, users made misspellings in the name of artist and/or song (to reduce the effect of misspellings, we used a phonetic search in the All Music database, so some common misspellings did not affect the classification). Finally, the All Music database is not complete, which is especially true in the case of classical music.

To evaluate the document classifier, we measured the number of incorrect classifications. We selected 200 random filenames and manually found the substyles to which they belong (occasionally using the All Music database and Google as an aid to find the substyles of obscure pieces). We then compared the manual classification with the one obtained from our document classifier. We considered a classification to be incorrect for a given document if the document classifier returned one or more substyles to which the document should not belong. Note that an "unknown" classification from our classifier, although very imprecise, is not incorrect as it would correspond to the root node of the classification hierarchy. In our evaluation, we found that 25% of the files were classified incorrectly.

However, a node can still be correctly classified even if some of its documents are misclassified. (If a node is properly classified, it will be possible to find the misclassified documents later on.) To evaluate the true effect of document misclassification, we selected 20 random nodes, we classified all their documents, and assigned the nodes to all the substyles of their respective documents. We considered a classification to be incorrect for a given *node* if the node was not assigned to one or more substyles to which the node should belong. In our evaluation, we found that only 4% of the nodes were classified incorrectly. This result shows that errors when classifying documents tend to cancel each other within a node. Specifically, even if we fail to classify a document as, for example, “Pop Rock,” it is likely that there will be some other “Pop Rock” document in the node that will be classified correctly so the node will still be assigned to the “Pop Rock” SON.

5.2 Evaluating our Query Classifier

For our experiments, queries were classified by hand by the authors of this paper. Queries were either classified in one or more substyles, a single style, or as “music”(the root of the hierarchy). In our experiments we used queries obtained from traces of actual queries sent to an OpenNap server run at Stanford [20]. Thus, by manually classifying queries, we are “guessing” what the users would have selected from say a drop-down menu as they submitted their queries.

Unfortunately, we cannot evaluate the correctness of the query classification method (we, of course, consider our classification of all queries to be correct). Nevertheless, we can study how precise our manual classification was (i.e., how many times queries were classified into a substyle, a style, or at the root of the classification hierarchy). We selected a trace of 50 *distinct* queries (the original query trace contained many duplicates which the authors of [20] believed were the result of cycles in the OpenNap overlay network) and then manually classified those queries. The result was that 8% of the queries were classified at the root of the hierarchy, 78% were classified a the style level of the hierarchy and 14% at the substyle level. As we will see in Section 7, the distribution of queries over hierarchy levels will impact the overall system performance, as more precisely classified queries can be executed more efficiently.

6 Nodes and SON Membership

In Section 3 we presented a conservative strategy for nodes to decide which SONs to join. Basically, under this strategy, nodes join all the SONs associated with a concept for which they have a document. This strategy guarantees that we will be able to find all the results, but it may increase both the number of nodes in each SON and the number of connections that a node needs to maintain. A less conservative strategy, where nodes join some of all the possible SONs, can have better performance. In the next subsection we introduce a non-conservative assignment strategy: Layered SONs.

6.1 Layered SONs

The Layered SONs approach exploits the very common zipfian data distribution in document storage systems. (It has been shown that the number of documents in a website when ranked in order of decreasing frequency, tend to be distributed according to Zipf’s Law [2].) For example, on the left side of Figure 8 we present a hypothetical histogram for a node with a zipfian data distribution (we’ll explain the rest of the figure shortly). In this histogram we can observe that 45% of the documents in the node belong to category c_1, about 35% of the documents belong to category c_2, while the remaining documents belong to categories c_3 to c_8. Thus, which SONs should the node join? The conservative strategy mandates that the node need to join SON_{c_1} through SON_{c_8}. However, if we assume that queries are uniform over all the documents in a category, it is clear that the node will have a higher probability of answering queries in SON_{c_1} and SON_{c_2} than queries in the other SONs. In other words, the benefit of having the node belong to SON_{c_1} and SON_{c_2} is high, while the benefit of joining the other SONs will be very small (and even negative due to the overhead of SONs). A very simple and aggressive alternative would be to have the node join only SON_{c_1} and SON_{c_2}. However, this alternative would prevent the system from finding the documents in the node that do not belong to categories c_1 and c_2.

Nodes determine which SONs to join based on the number of documents in each category. To illustrate, consider again Figure 8. At the right of the figure we present the hierarchy of concepts that will aid a node in deciding which SONs to join. In addition, a parameter of the Layered SON approach is the minimum percentage of documents that a node should have in a category to belong to the associated SON (alternatively, we can also use an absolute number of documents instead of a percentage). In the example, we have set that number at 15%. Let us now determine which SONs the node with the histogram at the left of Figure 8 should join. First, we consider all the base categories in the hierarchy tree (c_1 to c_8). As c_1 and c_2 are above 15%, the node joins SON_{c_1} and SON_{c_2}. As all the remaining categories are all below 15%, the node does not join their SONs. We then consider the second level categories (c_9, c_{10}, and c_{11}). As the combination of the non-assigned descendants of c_9, c_3 and c_4, is higher than 15%, the node joins SON_{c_9}. However, the node does not join the SON of c_{10} as the combination of c_5 and c_6 are not above 15%. Similarly the node does not join the SONs of c_{11} as c_7 and c_8 are below the threshold. Finally, the node joins the SON associated with the root of the tree ($SON_{c_{12}}$) as there were categories (c_5, c_6, c_7 and c_8) that are not part of any assignment. This final assignment is done regardless of the 15% threshold as this ensures that all documents in the node can be found (in our example, if we do not join $SON_{c_{12}}$ we will not be able to find the documents in the SONs of c_5, c_6, c_7 and c_8).

The conservative assignment is equivalent to a Layered SON where the threshold for joining a SON has been set to 0%. In this case, the node will join the SONs associated with all the base concepts for which it has one or more documents.

6.2 Experiments

In this subsection we contrast the result, in terms of SON size and number of SONs per node, of the conservative approach of Section 4.1 and the Layered SON approach. For reason of space, we will only consider the Style/Substyle classification hierarchy (the results for the other classification hierarchies are consistent with the ones presented here and in Section 4.1).

In Figure 9, we show the distribution of style SONs when using Layered SONs with a threshold of 35% and for the conservative assignment (labeled as 0% SON). The graphs do not include the "root" category to which, in practice, all nodes belong. ¿From the graph, we can see that 616 nodes (about 34% of the total nodes) need to belong to just one style. This result shows a significant improvement versus the conservative assignment of Section 4.1 when only 24% of the nodes belonged to one style. Moreover, 97% of the nodes need to belong to four or less style categories (versus 90% when doing conservative assignments).

Using layered SONs also helps reduce the number of nodes per SON. Figure 10 shows a histogram for the size of the SONs (excluding the "root" SON). ¿From the

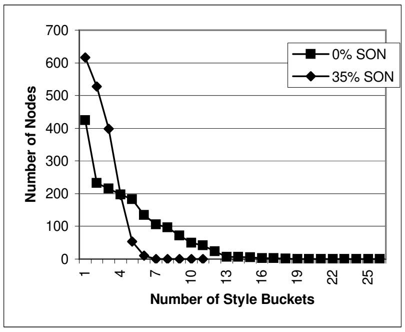


Fig. 9. Distribution of Style SONs

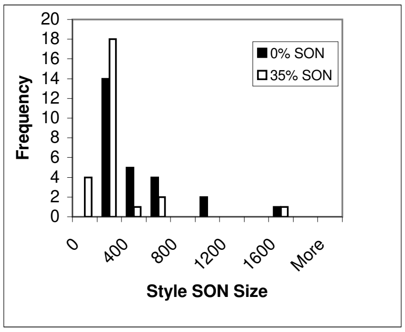


Fig. 10. SON Size Distribution for Style Hierarchy

graph we can see that by using Layered SONs we have a larger number of small SONs. However, as before, we still have a problem with the "Rock" style (rightmost bar in the graph) to which almost all nodes will have to belong. In conclusion, there is a significant reduction in the size of SONs when using Layered SONs instead of the conservative strategy. This reduction will lead to significant improvements in query performance.

We now consider Layered SONs based on the Style/Substyle classification hierarchy with a threshold of 10% (graph not shown). In this case, the conservative assignment strategy behave similarly in terms of the number of connections required at each node. However, the advantage of Layered SONs can be seen when considering the size of each SON as when using Layered SONs, SONs will have on average 135 nodes (versus 517 nodes for the conservative approach). Moreover, the Layered SON does not have any SONs with more than 875 nodes, while the conservative approach has 24. In conclusion, using Layered SONs with a Style/Substyle hierarchy produces a significant improvement versus the conservative assignment as we have much smaller SONs.

7 Searching SONs

In this section, we explore the problem of how to choose among a set of SONs when using Layered SONs. (We discussed in Section 3.1 the mechanisms used by nodes to actually send the queries to those SONs.)

7.1 Searching with Layered SONs

Searches in Layered SONs are done by first classifying the query. Then, the query is sent to the SON (or SONs) associated with the base concept (or concepts) of the query classification. Finally, the query is progressively sent higher up in the hierarchy until enough results are found. In case more than one concept is returned by the classifier, we do a sequential search in all the concepts returned before going higher up in the hierarchy. For example, when looking for a "Soft Rock" file we start with the nodes in the "Soft Rock" SON. If not enough results are found (recall that partial queries have a target number of results), we send the query to the "Rock" SON. Finally, if we still have not found enough results, we send the query to the "Music" SON. There are multiple approaches when searching with Layered SONs. In this paper we are concentrating on

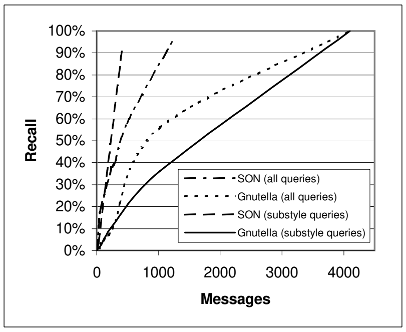


Fig. 11. Average number of Messages for a Query Trace

a single serial one (as our objective is to minimize number of messages). However, there are other approaches such as searching more than one SON in parallel (by asking each one for some fraction of the target results) which may result in higher number of messages, but will start producing results faster.

This search algorithm does not guarantee that all documents will be found if there are classification mistakes for documents. Not finding all documents may or may not be a problem depending on the P2P system, but in general, if we need to find all documents for a query (in the presence of classification mistakes), our only option is an exhaustive search among all nodes in the network. However, we will see that with our document classifier (which has an per-document classification mistake probability of 25%), we can find more than 95% of the documents that match a query. In addition, this search algorithm may result in duplicate results. Specifically, duplication can happen when a node belongs, at the same time, to a SON associated with a substyle and to the SON associated with the parent style of that substyle. In this case, a query that is sent to both SONs will search the node twice and thus it will find duplicate results.

7.2 Experiments

We will now consider two possible SON configurations and evaluate their performance against a Gnutella-like system. As before, we used the crawl of 1800 Napster nodes made at the University of Washington, which were classified using the All Music database. We assumed that the nodes in the network (both inside SONs and in the Gnutella network) were connected via an acyclic graph and that on average each node was connected to four other nodes. Although the assumption of an acyclic graph is not realistic, we are considering acyclic networks as the effect of cycles is independent of the creation of SONs. Cycles affect a P2P system by creating repeated messages containing queries that the receiving nodes have already seen. Therefore, an analysis of an acyclic P2P network gives us a lower estimate of the number of messages generated.

For this experiment we used 50 different random queries obtained from traces of actual queries sent to an OpenNap server run at Stanford [20]. These queries were classified by hand as described in Section 6. Queries classified at the substyle level were sent sequentially to the corresponding SON (or SONs), and then to the style-level SON. Queries classified at the style level, were first sent sequentially to all substyles of that style, and then to the style level. Queries classified at the root of the hierarchy were sent to all nodes. We measure the level of recall averaged for all 50 queries versus the number of messages sent in the system. As in the previous experiment, the graphs were obtained by running 50 simulations over randomly generated network topologies.

In Figure 11, we show the result of this experiment. The figure shows the number of messages sent versus the level of recall. Layered SONs were able to obtain the same level of matches with significantly fewer messages than the Gnutella-like system. However, Layered SONs do not achieve recall levels of 100% in general (average maximum recall was 93%) due to mistakes in the classification of nodes.

The results of Figure 11 show the average performance for all query types (dotted line). However, if a user is able to precisely classify his query, he will get significantly better performance. To illustrate this point, Figure 11 also shows with a dashed the number of messages sent versus the level of recall for queries classified at the substyle level (the lowest level of the hierarchy). In this case, we obtain a significant improvement versus Gnutella. For example, to obtain a recall level of 50%, Layered SONs required only 461 messages, while Gnutella needed 1731 messages, a reduction of 375% in the number of messages. Moreover, even at high recall levels, Layered SONs were able to reach a recall level of 92% with about 1/5 of the messages that Gnutella required.

The shape of the curve for the message performance of Gnutella is slightly different for all queries and for queries classified at the substyle level. The reason for this difference is very subtle. The authors of this paper were only able to classify very precisely (i.e. to the substyle level) queries for songs that are very well known. Due to their popularity, there are many copies of these songs throughout the network. Therefore, a Gnutella search approach will have a high probability of finding a match in many of the nodes visited, making the flooding of the network less of a problem than with more rare songs. Nevertheless, even in this case, Layered SONs performed much better than Gnutella.

8 Conclusion

We studied how to improve the efficiency of a peer-to-peer system by clustering nodes with similar content in Semantic Overlay Networks (SONs). We showed how SONs can efficiently process queries while preserving a high degree of node autonomy. We introduced Layered SONs, an approach that improves query performance even more at a cost of a slight reduction in the maximum achievable recall level. ¿From our experiments we conclude that SONs offer significant improvements versus random overlay networks, while keeping costs low. We believe that SONs, and in particular Layered SONs, can help improve the search performance of current and future P2P systems where data is naturally clustered.

References

1. A. Crespo and H. Garcia-Molina. Semantic overlay networks for p2p systems. Technical report, Stanford University, January 2003.
2. R. Korfhage. *Information storage and retrieval*. Wiley Computer Publishing, 1997.
3. D. Kossman. The state of the art in distributed queyr processing. *ACM Computing Survey*, September 2000.

4. J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *ASPLOS*, 2000.
5. C. Manning and H. Schutze. *Foundations of statistical natural language processing*. The MIT Press, 1999.
6. W. Nejdl, W. Siberski, M. Wolpers, and C. Schmitz. Routing and clustering in schema-based super peer networks.
7. B. R.-N. R. Baeza-Yates. *Modern Information Retrieval*. Addison Wesley, 1999.
8. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *ACM SIGCOMM*, 2001.
9. E. Rich and K. Knight. *Artificial Intelligence*. McGraw-Hill Inc., 1991.
10. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Middleware*, 2001.
11. M. Sahami and S. Y. M. Baldonado. Sonia: A service for organizing networked information autonomously. In *Proceedings of the Third ACM Conference on Digital Libraries*, 1998.
12. S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. Technical Report UW-CSE-01-06-02, University of Washington, 2002.
13. M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. A scalable and ontology-based p2p infrastructure for semantic web services.
14. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. ACM SIGCOMM*, 2001.
15. I. Witten and E. Frank. *Data Mining*. Morgan Kaufmann Publishers, 1999.
16. WWW: http://opennap.sourceforge.net. *OpenNap*.
17. WWW: http://vivisimo.com. *Vivisimo*.
18. WWW: http://www.allmusic.com. *All Music Guide*.
19. WWW: http://www.napster.com. *Napster*.
20. B. Yang and H. Garcia-Molina. Comparing hybrid peer-to-peer systems. In *Proceedings of the Twenty-first International Conference on Very Large Databases (VLDB'01)*, 2001.
21. B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, U. C. Berkeley, 2001.