Making Search Efficient on Gnutella-like P2P Systems

Yingwu Zhu
Department of ECECS
University of Cincinnati
zhuy@ececs.uc.edu

Xiaoyu Yang
Department of ECECS
University of Cincinnati
yangxu@ececs.uc.edu

Yiming Hu
Department of ECECS
University of Cincinnati
yhu@ececs.uc.edu

Abstract

Leveraging the state-of-the-art information retrieval (IR) algorithms like VSM and relevance ranking algorithm, we present GES, an efficient IR system built on top of Gnutella-like P2P networks. The key idea is that GES employs a distributed, content-based, and capacity-aware topology adaptation algorithm to organize nodes (each of which is represented by a node vector) into semantic groups. The intuition behind this design is that semantically associated nodes within a semantic group tend to be relevant to the same queries. Given a query, GES uses a capacity-aware search protocol based on semantic groups and selective one-hop node vector replication, to direct the query to the most relevant nodes which are responsible for the query, thereby achieving high recall with probing only a small faction of nodes. Moreover, GES adopts automatic query expansion techniques to improve quality of search results, and it is the first work to show that node vector size plays a very important role in system performance. The experimental results show that GES is very efficient, and even outperforms the centralized node clustering system like SETS.

1. Introduction

In the past years structured P2P systems [20, 16, 14, 23] have attracted tremendous attention from both the user and research communities. Such systems are adept at *exact-match* lookups: given a key, the system can locate the corresponding document within $O(\log N)$ hops (N is the number of nodes in the system). However, extending exact-match lookups to support complex queries (e.g., keyword search, semantic/content search) on structured P2P systems is non-trivial. A number of search techniques [9, 15, 22, 24] have been proposed on such systems. The main problem facing these search techniques is high maintenance cost in both overlay structure and document indices due to node churn in P2P networks. For example, the churn rate of coming and going nodes in a deployed P2P network (e.g., Gnutella) of $100,000$ nodes is expected to be over $1,600$ nodes per minute [5]. Node churn complicates the task of search on structured P2P networks. First, each node failure requires $O(\log N)$ repair operations to maintain the strictly structured overlay topology, e.g., to reconfigure the routing tables and thus preserve the efficiency and correctness of routing after each failure. Second, the lost documents and document indices have to be discovered and replicated to other nodes after failures to allow the aforementioned P2P search techniques to perform efficiently and effectively.

However, node churn causes little problem for Gnutella-like P2P systems because such systems employ an unstructured overlay. The unstructured overlay organizes nodes into a random graph and uses flooding or random walk [11] on the graph to retrieve relevant documents for a query. Each visited node evaluates the query locally on its own contents. Arbitrarily complex queries therefore can be easily supported on such unstructured systems, but the drawback is inefficiency — either a large portion of nodes have to be probed or some relevant documents have to be missed.

To improve search efficiency, a number of search techniques [11, 6, 7, 19, 2] have been proposed on Gnutella-like P2P systems. However, with very few exceptions [2], most of them ignore the state-of-the-art IR algorithms such as VSM (vector space model) and relevance ranking algorithms, and therefore may not be able to provide guarantee on *recall*. Worse yet, a query containing popular terms may return a superfluous number of documents beyond a user's capability to deal with.

In this paper we focus our study on a scenario where documents are organized into nodes according to the humans who created them, though the techniques we present here may be applicable to other scenarios such as interest-based locality [19] (i.e., organize nodes into semantic groups according to their interests). It is worth pointing out that we do not assume the documents on a node are restricted to only one single topic/area and they could be diverse instead (as will be shown in Section 5.3). The goal of our work is to improve the quality of search (e.g., high recall) while minimizing the associated cost (e.g., the number of nodes involved for a query). To achieve this goal, we introduce the concept of *semantic group*: nodes are organized into groups according to their *node vectors*. A *semantic link* connects two nodes if their node vectors are deemed to be relevant. Our design philosophy differs from existing work like SETS [2] in that we seek to improve search efficiency while retaining the simple, robust, and fully decentralized nature of Gnutella. Our design rationale is based on the intuition that semantically associated nodes within a semantic group tend to be relevant to the same queries. Given a query, we first locate the most relevant semantic groups for the query and then flood the query within the semantic groups for answers. In particular, we make the following contributions:

- Leveraging the IR algorithms, we propose a distributed, dynamic, and capacity-aware topology adaptation algorithm to organize nodes into semantic groups. Based on semantic groups and *selective one-hop node vector replication*, an efficient search protocol directs queries to the most relevant nodes which are responsible for the queries, thereby achieving high recall at very low cost.

- Our findings through detailed experiments suggest that an appropriate node vector size is a very good design choice in both search efficiency and effectiveness.

- We introduce automatic query expansion [12] into our system to improve the quality of search results in both recall and *precision*.

- Our results show that GES's capacity-aware mechanism can exploit node heterogeneity to further improve search efficiency.

The combination of the above techniques makes GES very efficient, and our experimental results have shown it even outperforms the centralized node clustering system like SETS [2].

The remainder of the paper is organized as follows. Section 2 gives an overview of related work. Section 3 provides necessary background. In Section 4, we describe the design of GES. In Section 5, we present our evaluation metrics, data, and simulation methodology. We provide our experimental results in Section 6. We finally conclude in Section 7.

2. Related Work

A number of P2P keyword search systems [15, 9, 21] have been built on top of structured P2P systems [20, 16, 14, 23]. All of them are based on global indexing where each node is responsible for the inverted list of some terms. Recently, some content-based P2P search systems [22, 24], leveraging the state-of-the-art IR algorithms, have been proposed on top of structured P2P systems. However, the main drawback of all these search systems is huge (overlay and document indexes) maintenance cost caused by node churn.

Centralized P2P systems such as Napster suffer from a single point of failure and performance bottleneck at the indexing servers. On the other hand, decentralized P2P systems like Gnutella, rely on flooding to answer queries, consuming huge amounts of bandwidth and CPU cycles and posing a scalability issue.

Improvements to Gnutella's flooding mechanism have been studied along three dimensions: random walks, guided search, and similar content group-based search. Lv et al. [11] proposed random walks in place of flooding to improve scalability. Crespo et al. [7] introduced the notion of "routing indices" to guide a query towards nodes which are more likely to have the requested documents. This search technique is similar to GES's biased walks which rely on replicated node vectors to direct a query towards most relevant nodes for answers. Systems such as [6, 19, 2, 13] organize nodes into either similar content groups or interest-shared groups to improve search efficiency on Gnutella-like P2P systems. However, with very few exceptions [2], most of them ignore the state-of-the-art IR algorithms such as VSM and advanced relevance ranking algorithms, and thus may not be able to provide guarantee on recall.

GES bears more similarity with SETS [2]. However, several important features distinguish GES from SETS. First, GES uses a distributed topology adaptation algorithm to organize nodes into semantic groups while in SETS a single designated node is responsible for clustering nodes into topic segments. Such a centralized structure may suffer from a single point of failure and performance bottleneck. Second, GES's topology adaptation and search protocol are capacity-aware, and it could exploit node capacity heterogeneity to significantly improve performance. Thirdly, GES is the first to explore the impact of node vector sizes on system performance, and show that an appropriate node vector size is very important in system design. Finally, GES is the first system to adopt automatic query expansion techniques into Gnutella-like P2P systems to improve the quality of search results.

GES's topology adaptation partly draws inspiration from Gia's [5]. However, GES's topology adaptation differs from Gia's in that it is mainly used to organize relevant nodes into same semantic groups to improve search efficiency while Gia uses topology adaptation to improve system scalability. Recent work [10] proposes a hybrid search technique on Gnutella-like systems in which structured P2P search techniques are used to index and locate rare documents and flooding techniques are used to locate highly replicated documents.

3. Background: Vector Space Model (VSM)

In VSM [3] each document/query is represented by a vector of terms. The terms are stemmed words [1] which occur within the document/query. In addition, stop words [2] and highly frequent words are removed from the term vector. Each term in the vector is assigned a weight. Terms with a relatively heavy weight are generally deemed to be central to a document. To evaluate whether a document is relevant to a query, VSM measures the relevance between the query vector and the document vector. Typically, for a document D and a query Q (suppose the term vectors of D and Q have been already normalized) the relevance score is computed as:

$$REL(D, Q) = \sum_{t \in D,Q} d_t \cdot q_t \tag{1}$$

Where t is a term appearing in both D and Q, q_t is term t's weight in query Q, and d_t is term t's weight in document D. Documents with high relevance score are identified as search results for a query.

A number of term weighting schemes have been proposed for VSM, among which *tf-idf* is a scheme in which the weight of a term is assigned a high numeric value if the term is frequent in a document but infrequent in other documents. The main drawback of *tf-idf* is that it requires some *global* information (i.e., the document frequency *df*, which represents the number of documents where a term occurs) to compute a term's weight. In our design, we use a "dampened" *tf* scheme, where each term t is assigned a weight in the form of $d_t = 1 + \log f_t$ (f_t is t's term frequency in a document). Previous work [18] has shown that this scheme can produce high quality document clusters without requiring the global information.

4. System Design

In this section, we provide an overview of GES, describe its node vectors, discuss its topology adaptation algorithm, present its selective one-hop node vector replication, and detail its search protocol.

4.1. Overview

In GES, each node has a node vector, a compact summary of its documents. A node may have two types of links/connections, namely *random links* and *semantic links*. Random links connect irrelevant nodes while semantic links organize relevant nodes into semantic groups. Maintaining random links on each node helps to efficiently discover different semantic groups. The topology adaptation algorithm is first performed to connect a node to the rest of the network through random links and semantic links [3]. Given a query, the search protocol, which is based on semantic groups and selective one-hop node vector replication, directs the query through random links by biased walks to the most relevant semantic groups and then floods the query through semantic links within the semantic groups to retrieve relevant documents.

4.2. Node Vectors

To determine whether two nodes are relevant or not, we introduce the concept of *node vector*. A node vector characterizes a node's documents. We derive a node X's node vector from its documents as follows. First, each of X's documents is represented by a *temporary* term vector where each term t's weight is represented by its term frequency f_t. Second, all temporary term vectors of X's documents are summed up and we get a new vector in which each term component t has a weight f'_t. For each term t, we replace its weight f'_t with $1 + \log f'_t$. Finally, we normalize the new vector and the normalized vector is called X's node vector.

Given two nodes X and Y, their relevance score is computed as:

$$REL(X, Y) = \sum_{t \in X,Y} w_{X,t} \cdot w_{Y,t} \tag{2}$$

Where t is a term appearing in both X and Y, $w_{X,t}$ is term t's weight in X, and $w_{Y,t}$ is term t's weight in Y. If the relevance score is less than certain relevance threshold, nodes X and Y are deemed to be irrelevant. Otherwise, they are deemed to be relevant.

Node vectors are also used to determine the relevance of a node X and a query Q according to Equation 3, as will be shown later in biased walks during search.

$$REL(X, Q) = \sum_{t \in X,Q} w_{X,t} \cdot w_{Q,t} \tag{3}$$

4.3. Topology Adaptation Algorithm

The task of topology adaptation algorithm is to connect a node to the rest of the network, and more importantly, to connect a node to a semantic group (if it can find one) through semantic links. When a node joins the system, it first uses a bootstrapping mechanism in Gnutella to connect to the rest of the network. Initially, it may not have any information about other nodes' contents (i.e., node vectors) as

[1]Stemmed words are the words that provide the root for other related words. E.g., the stemmed word for words *restarted*, *restarts*, and *restarting* is *restart* by removing their suffixes.

[2]Stop words are those words that are considered non-informative, like function words *of, the, a*, etc.

[3]If a node cannot find a relevant node, it connects itself to the rest of the network *only* through random links.

well as semantic groups. Its attempt to gain such information is performed through *random walks*. A random walk is a well-known technique in which a query message is forwarded to a randomly chosen neighbor at each step until sufficient responses are found. In our system, the duration of a random walk is bound by a TTL (time-to-live).

A random walk query message contains a node's node vector, a relevance threshold *REL_THRESHOLD*, the maximum number of responses *MAX_RESPONSES*, and TTL. The random walk returns a set of nodes. In GES, a node actually periodically issues two queries, one requesting nodes whose relevance is lower than *REL_THRESHOLD*, and the other requesting nodes whose relevance is higher than or equal to *REL_THRESHOLD*. Note that the relevance score is computed using Equation 2. The returned nodes are added to the query initiator node's two *host caches*: *random host cache* and *semantic host cache*, respectively, according to their relevance scores. Each entry of host caches consists of *a node's IP address, port number, node capacity, node degree, node vector* [4], and *relevance score* (keeping pre-computed relevance scores in cache avoids recomputing). These two caches are maintained throughout the lifetime of the node. Moreover, each cache has a size constraint and uses FIFO as replacement strategy.

Each node periodically checks its two caches for random and semantic neighbor addition/replacement. To add a new *semantic neighbor* (a neighbor node connected by a semantic link), a node (say X) chooses a node from its semantic cache that is not dead and not already a neighbor and with the *highest* relevance score. Node X then uses a three-way handshake protocol to the chosen neighbor candidate, say Y. During handshake, each node decides independently whether or not to accept the other node as a new semantic neighbor based upon its own *MAX_SEM_LINKS* (the maximum number of semantic neighbors), *SEM_LINKS* (the number of current semantic neighbors), and the new node (i.e., the relevance score). If *SEM_LINKS* is less than *MAX_SEM_LINKS*, the node automatically accepts this new connection. Otherwise, the node must check if it can find an appropriate semantic neighbor to drop and replace with the new neighbor candidate. X always accepts Y and drops an existing semantic neighbor if Y's relevance score is higher than *all* of X's current semantic neighbors. Otherwise, it makes a decision whether to accept Y or not as follows. From all of X's semantic neighbors whose relevance scores are lower than that of Y and which are not poorly connected [5], we choose the neighbor Z which has the lowest relevance score. Then X drops Z and add Y into its semantic neighbors.

[4]The semantic host cache does not contain node vectors.

[5]As will be shown in section 5, each node has a minimum degree constraint, and a typical value is 3. If a node's degree is less than or equal to the minimum constraint value, this node is identified as a poorly-connected node.

To add a new *random neighbor* (a neighbor node connected by a random link), node X chooses a node from its random cache which is not dead and not already a neighbor and has a capacity greater than its own capacity (e.g., the highest capacity node is preferred). If no such candidate node exists, X randomly chooses a node. X then initiates a three-way handshake to the chosen random neighbor candidate, say Y. During the handshake, each node independently decides whether or not to accept the other node as a new random neighbor upon the capacities and degrees of its existing random neighbors and the new node. If X's *RND_LINKS* (the number of random neighbors) is less than *MAX_RND_LINKS* (the maximum number of random neighbors), the node automatically accepts this new node as a random neighbor. Otherwise, the node must check if it can find an appropriate random neighbor to drop and replace with the new node. X always drops an existing random neighbor in favor of Y if Y has capacity higher than *all* of X's existing random neighbors. Otherwise, it decides whether to accept Y or not as follows. From all of X's random neighbors that have capacity less than or equal to that of Y, we choose the neighbor Z which has the highest degree. Z will be dropped and replaced with Y *only* if Y has lower degree than that of Z. This prevents us from dropping already poorly-connected neighbors and isolating them from the rest of the network.

Discussion. The goal of the topology adaptation is to ensure that: (1) relevant nodes are organized into semantic groups through semantic links, by periodically issuing random walk queries for semantic neighbors and performing semantic neighbor addition/replacement, and (2) high capacity nodes have high degree and low capacity nodes are within short reach of higher capacity nodes, by periodically issuing random walk queries for random neighbors and performing random neighbor addition/replacement. Note that in GES a *direct* semantic link connects two *most* relevant nodes while in other systems like SETS a *local link* does not necessarily mean that two connected node are most relevant within a topic segment.

For random walk queries requesting relevant nodes, we can actually do optimizations as follows. When a query arrives at a node (say Y) which is deemed to be relevant to the initiator node (say X), Y can first choose some relevant nodes from its semantic host cache for responses with some probability. If the *MAX_RESPONSES* has been reached, the query reply is routed back to X. Otherwise, Y biased walks the query through one of its semantic links with some probability. Further, relevant nodes within a semantic group can exchange the content of their host caches. Currently GES does not adopt such optimizations.

Each node also continuously keeps track of the relevance scores of both semantic links and random links. If the relevance score of a semantic link drops below

SEM_THRESHOLD due to dynamically changing documents in either node (and thus changing node vectors), we simply drop the semantic link and add the neighbor information into the random host cache. Similarly, if the relevance score of a random link becomes above *SEM_THRESHOLD*, we simply drop the random link and add the neighbor information into its semantic host cache. As a result, the topology adaptation process performed thereafter can adapt to dynamically changing node vectors of each node's existing neighbors.

4.4. Selective One-hop Node Vector Replication

To improve search efficiency, each node maintains the node vectors of *all* of its random neighbors in memory. Note that we do not maintain those of its semantic neighbors. This is why we call it *selective* replication. When a random connection is lost, either because the random neighbor node leaves the system, or due to topology adaptation, the node vector for this neighbor gets flushed from memory. A node periodically checks the replicated node vectors through heartbeat messages with each random neighbor in case that a neighbor node may add/remove documents. This allows nodes to adapt to dynamically changing node vectors (due to dynamically changing of documents) and keep replicated node vectors up-to-date and consistent.

4.5. Search Protocol

The combination of topology adaptation algorithm and selective one-hop node vector replication has paved the way for our content-based, capacity-aware search protocol. We first present the search protocol while leaving its capacity-aware mechanism for discussion later in this section.

Given a query, the search protocol is performed as follows. First, GES uses *biased walks*, rather than random walks, to forward the query through *random links*. During biased walks, each node along the route looks up its locally stored documents for those satisfying the query: each document is evaluated using Equation 1 and a relevance score is computed [6]; if the relevance score is higher than or equal to certain relevance threshold, this document is identified as a relevant document for the query. If any such a relevant document is identified, then the node (say X) is called a *semantic group target node*, where the query terminates biased walks and starts flooding. Otherwise, X selects a neighbor (say Y) from its *random neighbors* whose node vector is *most* relevant to the query vector according to Equation 3, and forwards the query to Y. The biased walks are repeated until a semantic group target node is identified.

[6]We may narrow down the evaluation scope by clustering a node's documents first.

The target node then floods the query along *all* of its *semantic links*: each semantic neighbor evaluates the query against its documents and then floods the query along its own semantic links. During flooding, we can allow the query to probe all of the nodes within a semantic group or only a fraction of nodes by imposing a flooding radius constraint from the target node (called *controlled flooding*). The relevant documents found within the semantic group are directly reported to the target node. Note that each query contains a $MAX\_RESPONSES$ parameter. The target node aggregates the relevant documents, reports them directly to the query initiator node (which will present highest relevance ranking documents to the user), and decreases $MAX\_RESPONSES$ by the number of relevant documents. If $MAX\_RESPONSES$ becomes less than or equal to zero, the query is simply discarded. Otherwise, the query starts biased walks from the target node again and repeats the above search process until sufficient responses are found.

During both biased walks and flooding, we use book-keeping techniques to sidestep redundant paths. In GES, each query is assigned a unique identifier $GUID$ by its initiator node. Each node keeps track of the neighbors to which it has already forwarded the query with the same $GUID$. During biased walks, if a query with the same $GUID$ arrives back at a node (say X), it is forwarded to a different random neighbor with the highest relevance score among those random neighbors to which X has not forwarded the query yet. This reduces the probability that a query traverses the same link twice. However, to ensure forward progress, if X has already sent the query to all of its random neighbors, it flushes the book-keeping state and starts reusing its random neighbors. On the other hand, during flooding, if a query with the same $GUID$ arrives back at the node, the query is simply discarded. Note that GES nodes treat query messages with the same $GUID$s differently during biased walks and flooding.

The search protocol we discussed above does not consider node capacity heterogeneity. Now we incorporate the capacity-aware mechanism into the search protocol to make search more efficient in a system where node capacities are heterogeneous. Due to the fact that the topology adaptation algorithm takes into account the heterogeneity of node capacities *only* in random link construction, the search protocol only needs to adapt its biased walks while retaining the flooding part untouched. During biased walks for a query, each node makes a decision how to forward the query based on *the query vector, its own capacity, the capacities of all of its random neighbors, and the node vectors of all of its random neighbors*. If the node (say X) is a supernode (whose capacity is higher than certain threshold), it forwards the query to a random neighbor whose node vector is most relevant to the query vector. Otherwise, X must check all of its random neighbors and chooses a neighbor (say Y) with the highest capacity. If Y is a supernode, X forwards the query to Y, hoping that high capacity nodes can typically provide useful information for the query. Otherwise, X forwards the query to a random neighbor whose node vector is most relevant to the query vector. The biased walks are repeated until a target node is reached.

Discussion. In summary, query flooding within a semantic group is based on the intuition that semantically associated nodes tend to be relevant to the same queries and can provide useful responses for them. The biased walks, taking advantage of the heterogeneity of node capacities and selective one-hop node vector replication, forward a query either to a supernode neighbor with the hope that high capacity nodes can typically provide useful information for the query, or to the *most* relevant random neighbor if no such a supernode exists. In our system, in addition to $MAX\_RESPONSES$, each query is also bound by a TTL parameter. Note that flooding within semantic groups keeps us from exactly keeping track of the TTL. For simplicity, GES decreases the TTL by one at each step *only* during biased walks. Once TTL hits zero, the query message is dropped and no longer forwarded.

5. Performance Evaluation

In this section we discuss the design of experiments to evaluate the effectiveness and efficiency of GES. We start by giving a brief overview of two other unstructured P2P search systems. We then discuss performance metrics and data for our evaluation. Finally, we describe our methodology.

5.1. Random and SETS

Random: Search using random walks over Gnutella-like P2P systems. This represents the recommended search technique proposed in [11]. It is used to address the scalability problem posed by flooding on Gnutella-like P2P systems. A random walk is essentially a blind search in that at each step a query is forwarded to a randomly chosen node without considering any hint of how likely the next node will have answers for the query.

SETS: Search using a topic-driven query routing protocol on a topic-segmented overlay built from Gnutella-like P2P systems [2]. A topic segment in SETS is similar to a semantic group in GES. However, SETS differs from GES in that the topic-segmented overlay is constructed by performing node clustering at a *single* designated node, and each cluster corresponds to a topic segment. Given a query, SETS first computes R topic segments which are most relevant to the query and then routes the query to these segments for relevant documents. When a node joins the system, it first has to contact the designated node for the information about all the C topic segments and then joins the most relevant segment. Moreover, as nodes join/leave or their document collections change, the designated node has to recompute topic segments to keep them up-to-date and then disseminates them throughout the system. This centralized structure could be a single point of failure and performance bottleneck.

5.2. Performance Metrics

The metrics we used to express the benefits and cost of our design are:

- *Recall*: It is a main metric used to quantify the quality of search results, and is defined as the number of retrieved relevant documents divided by the number of relevant documents.

- *Precision@r*: It is another metric used to quantify the quality of search results, and is defined as the number of retrieved relevant documents divided by the number r of retrieved documents. We are particularly interested in high-end precision (e.g., prec@15) because a recent study [8] has shown that users only view top 10 search results.

- *Query Processing Cost*: It is defined as the fraction of nodes in the system which are involved in a query processing. A lower query processing cost increases system scalability since system resource consumption is proportional to the number of nodes visited by a query. In addition, query processing cost measures the quality of topology adaptation. If the quality of topology adaptation is poor, a query may probe many irrelevant nodes, thereby increasing query processing cost.

5.3. Data

The data we used to evaluate our design is *TREC-1,2-AP* [1]. The TREC corpus is a standard benchmark widely used in the IR community. *TREC-1,2-AP* contains AP Newswire documents in TREC CDs 1 and 2. We extracted those documents with text and valid author fields from this original document collection. We assumed that each author corresponds to a node and his/her associated documents are stored on the corresponding node. This resulted in $80,008$ documents distributed over 1880 nodes. The mean, 1th-percentile and 99th-percentile of the number of documents per node are 42.5, 1 and 417, respectively. The term vector of a document was derived from the text field using VSM. The terms in each document vector were stemmed. We also used a list of 571 stop words from SMART [4] to remove stop words from each document vector. Each document vector on average had 179 unique terms.

The queries we used are from TREC-3 ad hoc topics (151-200). The query vector was derived from the *title* field using VSM. The terms in each query vector were stemmed and stop words were removed as well. Thus, these 50 queries each had on average 3.5 unique terms. Moreover, the 50 queries each comes with a query relevant judgment file which contains a set of already identified relevant documents (by TREC-3 ad hoc query assessors). Since we only used 80, 008 AP Newswire documents with valid author and text fields, we removed those documents which do not appear in this 80, 008 document set from the accompanying relevant judgment files.

Our preliminary analysis of the TREC data shows that more than 50% nodes provide relevant documents for two or more queries (the maximum is 12 queries). We found that the topics of these queries (151-200) are very different. We conclude that documents created by an author (and thus distributed on a node) are not restricted to only one single topic/area. Hence, the TREC data we used in our evaluation does not assume a node specializes in one single topic, and a very large portion of nodes hold diverse documents indeed (the checking of documents confirmed this).

5.4. Methodology

GES simulations started with a randomly-connected topology, and then used topology adaptation algorithm to restructure the initial topology. GES's topology adaptation algorithm uses four preconfigured parameters: min_links, max_links, α, and $node\_rel\_threshold$. min_links is the minimum number of neighbors per node, and we used $min\_links = 3$ throughout all of our experiments. max_links is the maximum number of neighbors per node, we set $max\_links = 8$ in the experiments where node capacities are assumed to be uniform. In the experiments where node capacities are heterogeneous, we set max_links to 128. However, there is a constraint on max_links, i.e., $max\_links = min(max\_links, \lfloor \frac{C}{min\_unit} \rfloor)$, where C is a node's capacity and min_unit represents the finest level of granularity into which a node's capacity is split. Our preliminary results suggested $min\_unit = 4$ is a good design choice. α represents the maximum fraction of max_links devoted to semantic links, and our preliminary results suggested $\alpha = 0.5$ is a good design choice. $node\_rel\_threshold$ represents the node relevance threshold we used to perform topology adaptation and our preliminary results showed that 0.45 is a good choice.

For SETS, a designated node performed topic segmentation to reconfigure the initial randomly-connected topology. Each node had 4 long-distance links and 4 local links. The number of topic segments are 256, which represents the best case for performance compared to 32, 64, and 128 topic segments. For Random, there is no topology adaptation, and it used a random graph. Throughout all of our experiments, we chose uniformly random graphs with an average degree of 8. The primary purpose of choosing such uniformly random graphs is to sidestep unnecessarily biasing against SETS and Random.

In most of our experiments, we assumed node capacities are uniform by default (unless otherwise specified). For the experiments where node capacities are heterogeneous, node capacities are assigned based on a Gnutella-like profile [17]. We assigned capacity of $1\times$, $10\times$, $10^2\times$, $10^3\times$ and $10^4\times$ to nodes with probability of 20%, 45%, 30%, 4.9% and 0.1%, respectively. Nodes with capacities $10^3\times$ and $10^4\times$ were assumed to be supernodes.

6. Experimental Results

6.1 Performance Comparison

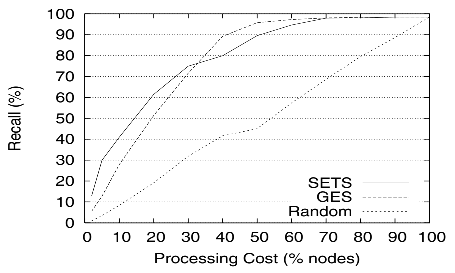

Figure 1. Recall vs. processing cost.

We present evaluation results comparing the performance of GES against SETS and Random, as shown in Figure 1. Several observations can be drawn from this figure: (1) GES and SETS outperform Random substantially, achieving higher recall at lower query processing cost. (2) Compared to GES, SETS achieves higher recall when probing less than 30% nodes. This is explained by the fact that SETS takes advantage of knowing the global C (=256) topic segments and therefore can quickly and precisely locate the most relevant topic segments to look up relevant documents. GES, instead has to use biased walks to locate a target node, and then floods the query within the corresponding semantic group for relevant documents. If the target node is not a right one (which actually does not contain relevant documents, though some of its documents have relevance score high enough to be deemed relevant), some irrelevant nodes are unavoidably probed. The overhead of locating a right target node hurts the performance of GES, especially when probing *only* a small fraction of nodes. However, GES still achieves about 71.6% recall by probing *only* 30% nodes.

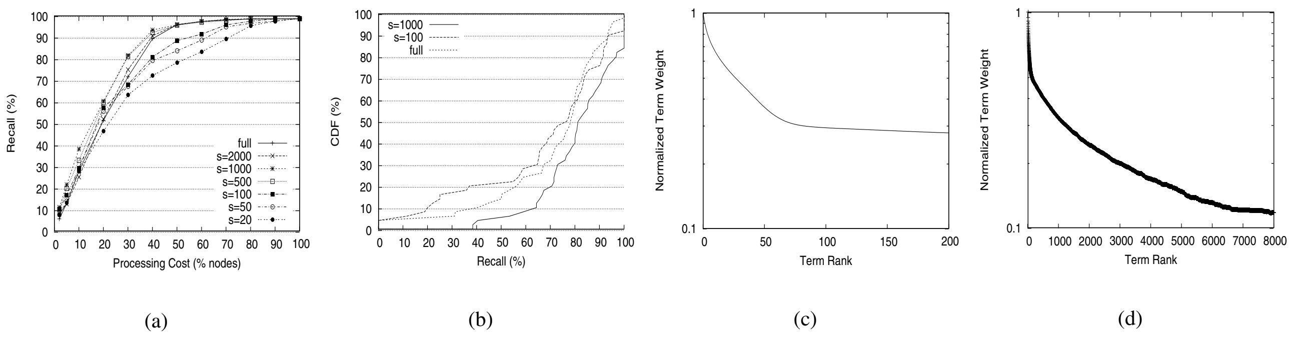

Figure 2. (a) The performance of various node vector sizes (s represents the size of node vectors and "full" means using full-size node vectors). (b) Cumulative distribution of recall. (c) Ranked term weight for the TREC documents, normalized to the biggest term weight in each document. (d) Ranked term weight for the node vectors (top 8000 terms), normalized to the biggest term weight in each node vector.

(3) GES outperforms SETS when exploring more than 30% of the network. It achieves 89.3% recall by visiting 40% nodes while SETS achieves 80% recall in this case. We give the following explanations. First, the overhead of locating a right target node (and thus the semantic group) is amortized by probing more nodes. Secondly, the nature of GES's topology adaptation connects the *most* relevant nodes through *direct* semantic links, and it ensures a query probes the *most* relevant nodes first along semantic links. However, SETS does not distinguish the relevance between nodes within a topic segment and local links do not necessarily reflect that the *most* relevant nodes have *direct* connections. Therefore some irrelevant nodes within topic segments are unavoidably visited when flooding the query within topic segments. (4) When exploring the whole network, the recall achieved by all three systems is 98.5%. This is because queries are short on average with only 3.5 terms in our experiments. Some relevant documents could not be identified because their relevance scores computed using Equation 1 are 0. During query evaluation, they are mistakenly deemed to be irrelevant due to such a low relevance score. In other words, with such short queries, the maximum recall achieved by a centralized IR system is 98.5%.

6.2. Effect of Node Vector Sizes

In our experiments the average node vector size is 1776, the 1th-percentile of the node vector size is 88, and the 99th-percentile of the node vector size is 8099. In this section, we explore the effect of node vector sizes by conducting experiments to provide insight on the following questions:

- What is a good node vector size in GES's performance, i.e., recall?
- How could a substantial reduction in node vector size affect performance, i.e., recall?

To the best of our knowledge, this is the first work to explore the effect of node vector sizes and SETS uses only full-size node vectors. Figure 2(a) depicts the recall vs. processing cost for various node vector sizes. Several observations can be drawn from this figure: (1) The node vector sizes of 1000 and 500 perform the best [7], achieving 81% recall when probing only 30% nodes. (2) The node vector size of 100 still works very well, achieving about 68% recall when probing 30% nodes. (3) The node vector sizes of 20 and 50, representing the substantial reduction in node vector size, perform surprisingly well, achieving 44-55% and 63-67% recall when probing 20% and 30% nodes, respectively. Figure 2(b) plots the cumulative distribution of recall for the queries with respect to node vector size when probing 30% nodes (we observed similar characteristics on other cases). Note that the node vector size of 1000 outperforms other node vector sizes significantly.

From the above observations, a question is naturally raised: why does an appropriate node vector size (e.g., 1000) perform the best while both a substantially larger (even full) node vector size and a substantially smaller one hurt the performance? To answer this question, we now look at Figure 2(c) and (d). As shown in Figure 2(c), the weight of the top 50 terms drops very fast. This confirms our intuition that a small number of terms (e.g.,50) are central to a document. They are capable of characterizing a document like the "Keyword" section of a technical paper. As shown in Figure 2(d), the weight of the top 100 terms drops faster than Zipf distribution, and the weight of top 1000 terms also drops very fast. This shows that a relatively small number of terms are capable of characterizing a node's contents, due to the fact that a node vector is derived from its documents each of which is characterized by

[7]The node vector size of 1000 means we use top 1000 terms of a node vector.

	processing cost (% nodes)						
	2%	*5%*	*10%*	*20%*	*30%*	*40%*	*≥50%*
GES(1000+heter):SETS	63.8%	8.3%	16.1%	17.9%	13.3%	18.5%	≤7.4%

Table 1. The recall improvements with respect to query processing cost made by GES(1000+heter) on SETS. GES(1000+heter) represents GES which uses the node vector size of 1000 and considers heterogeneity.

a small number of top terms (with heaviest weight).

Therefore, a substantially reduction in the node vector size (e.g., 20 or 50) still performs very well since the node relevance score (according to Equation 2) is mostly determined by the top terms. However, a substantially small node vector size also loses many important terms, and this hurts the quality of topology adaptation which fails to identify some relevant nodes and thus is unable to put some actually relevant nodes within a semantic group, thereby impairing search performance. On the other hand, a substantially large node vector size (2000, or even full size) contains too many (tens, and even hundreds of) unimportant terms which interfere with the computation of the relevance score in Equation 2. Sometimes two nodes may be irrelevant even if their relevance score is high according to Equation 2. We illustrate such interference through an example. Suppose two node vectors X and Y. They do not share top terms (say, 100 or 500), but they share many (say, tens or hundreds of) unimportant terms. According to Equation 2, their relevance score may be high enough to be mistakenly deemed to be relevant. This will negatively affect the quality of topology adaptation such that topology adaptation could cluster irrelevant nodes into a semantic group. The node vector size of 1000 strikes the balance. On the one hand, it catches most (or all) of the top terms, and on the other hand, it reduces the negative impact of tens or even hundreds of unimportant terms, therefore achieving the best performance.

An appropriate node vector size (e.g., 1000) outperforms both a substantially larger (and full) node vector size and a substantially smaller node vector size. Moreover, compared to a larger node vector size, it also brings down the costs in *memory consumption (selective one-hop node vector replication, and random host cache), bandwidth usage (reduced message size during topology adaptation), and computation (the relevance score calculation consumes more time for larger node vector sizes)*. A significantly small node vector size (e.g., 50) can further bring down the aforementioned costs, while still showing pretty good performance. As a result, the node vector size exhibits a good design tradeoff in our system. We leave the issue of how to dynamically determine an appropriate node vector size (e.g., through sampling) to our future work.

6.3. Other Results

Due to space constraints, we briefly report some results in this section. Please refer to our technical report [25] for details.

Automatic Query Expansion. Automatic query expansion (or automatic relevance feedback) [12] has been shown to be a very effective technique to improve precision and recall in centralized IR systems, especially for short queries. Given a query, GES first retrieves a small number of most relevant documents which serve as *feedback documents*. GES then chooses certain number of top terms in the feedback documents and adds them to the initial query. The new query is used to retrieve the final set of documents. Our experimental results show that automatic query expansion improves precision, but not significantly. For example, when the number of added terms is 30, the improvement on precision@15 is about 10%. Automatic query expansion also improves recall by about 26% (the number of added terms is 30).

Impact of Heterogeneity. Table 1 summarizes the recall improvements made by GES(1000+heter) on SETS which does not consider capacity heterogeneity in its design.

7. Conclusions and Future Work

In this paper, we present GES, an architecture for P2P information retrieval on Gnutella while retaining the simple, robust, and fully decentralized nature of Gnutella. GES differs from existing similar content group-based search techniques in some or all of the following aspects: (1) it uses a fully distributed and dynamic topology adaptation algorithm to form semantic groups; (2) it leverages the state-of-the-art IR algorithms like VSM and relevance ranking algorithms, thereby improving recall and precision on queries; (3) its capacity-aware mechanism allows it to exploit node heterogeneity to further improve performance; (4) it is the first to show that the node vector size plays a very important role in system performance; (5) it employs automatic query expansion to improve the quality of search results. We have shown that the combination of our proposed techniques results in a P2P search system which is efficient and even outperforms the centralized node clustering system like SETS.

Our future work includes studying the issue of how to determine a node's satisfaction degree in topology adaptation, and obtaining larger data corpus to experiment on GES's scalability (involving more nodes) though we believe GES's efficient search could make it scalable. We also notice that documents on a node could be diverse, and we need to distinguish diverse topics in a node's documents for better semantic group formation and thus better search performance. To handle such diverse topics in a node, we could introduce a notion of "virtual node". A node with diverse topic documents could locally cluster its documents using data clustering techniques and each cluster corresponds to a virtual node. A node could host multiple virtual nodes, each of which independently participates in GES's topology adaptation and search protocol.

8. Acknowledgments

This work is supported in part by National Science Foundation under Career Award CCR-9984852 and ACI-0232647, and the Ohio Board of Regents.

References

[1] Text retrieval conference (trec). http://trec.nist.org.

[2] M. Bawa, G. Manku, and P. Raghavan. SETS: Search enhanced by topic segmentation. In *Proceedings of The 26th Annual International ACM SIGIR Conference*, pages 306–313, Toronto, Canada, July 2003.

[3] M. W. Berry, Z. Drmac, and E. R. Jessup. Matrices, vector spaces, and information retrieval. *SIAM Review*, 41(2):335–362, 1999.

[4] C. Buckley. Implementation of the smart information retrieval system. Technical Report TR85-686, Department of Computer Science, Cornell University, May 1985.

[5] Y. Chawathe, S. Ratnasamy, L. Breslau, and N. Lanham. Making Gnutella-like P2P systems scalable. In *Proceedings of ACM SIGCOMM*, pages 407–418, Karlsruhe, Germany, Aug. 2003.

[6] E. Cohen, H. Kaplan, and A. Fiat. Associative search in peer to peer networks: Harnessing latent semantics. In *Proceedings of IEEE INFOCOM*, volume 2, pages 1261–1271, San Francisco, CA, Apr. 2003.

[7] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 23–32, Vienna, Austria, July 2002.

[8] R. Lempel and S. Moran. Optimizing result prefetching in web search engines with segmented indices. In *Proceedings of VLDB*, 2001.

[9] J. Li, B. T. Loo, J. Hellerstein, F. Kaashoek, D. R. Karger, and R. Morris. On the feasibility of peer-to-peer web indexing and search. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems(IPTPS)*, pages 207–215, Berkeley, CA, Feb. 2003.

[10] B. T. Loo, R. Huebsch, I. Stoica, and J. M. Hellerstein. The case for a hybrid P2P search infrastructure. In *Proceedings of 3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, San Diego, CA, Feb. 2004.

[11] Q. Lv, P. Cao, and E. Cohen. Search and replication in unstructured peer-to-peer networks. In *Proceedings of 16th ACM Annual International Conference on Supercomputing (ICS)*, pages 84–95, New York, NY, June 2002.

[12] M. Mitra, A. Singhal, and C. Buckley. Improving automatic query expansion. In *Proceedings of ACM SIGIR*, pages 206–214, Melbourne, Australia, 1998.

[13] C. H. Ng and K. C. Sia. Peer clustering and firework query model. In *Proceedings of WWW*, 2002.

[14] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, pages 161–172, San Diego, CA, Aug. 2001.

[15] P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *Proceedings of ACM/IFIP/USENIX International Middleware Conference (Middleware)*, pages 21–40, Rio de Janeiro, Brazil, June 2003.

[16] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed System Platforms (Middleware)*, pages 329–350, Heidelberg, Germany, Nov. 2001.

[17] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking(MMCN)*, San Jose, CA, Jan. 2002.

[18] H. Schutze and C. Silverstein. A comparison of projections for efficient document clustering. In *Proceedings of ACM SIGIR*, pages 74–81, Philadelphia, PA, July 1997.

[19] K. Spripanidkulchai, B. Maggs, and H. Zhang. Efficient content location using interest-based locality in peer-to-peer systems. In *Proceedings of IEEE INFOCOM*, volume 3, pages 2166–2176, San Francisco, CA, Mar. 2003.

[20] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, pages 149–160, San Diego, CA, Aug. 2001.

[21] C. Tang and S. Dwarkadas. Hybrid global-local indexing for efficient peer-to-peer information retrieval. In *Proceedings of USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, Mar. 2004.

[22] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proceedings of ACM SIGCOMM*, pages 175–186, Karlsruhe, Germany, Aug. 2003.

[23] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerance wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, University of California, Berkeley, Apr. 2001.

[24] Y. Zhu, H. Wang, and Y. Hu. Integrating semantics-based access mechanisms with P2P file systems. In *Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, pages 118–125, Linkping, Sweden, Sept. 2003.

[25] Y. Zhu, X. Yang, and Y. Hu. Making search efficient on Gnutella-like P2P systems. Technical Report TR-270/01/04/ECECS, University of Cincinnati, Jan. 2004.