The Case for a Hybrid P2P Search Infrastructure

Boon Thau Loo* Ryan Huebsch* Ion Stoica* Joseph M. Hellerstein*†

*University of California at Berkeley †Intel Research Berkeley
{boonloo, huebsch, istoica, jmh}@cs.berkeley.edu

Abstract

Popular P2P file-sharing systems like Gnutella and Kazaa use unstructured network designs. These networks typically adopt flooding-based search techniques to locate files. While flooding-based techniques are effective for locating highly replicated items, they are poorly suited for locating rare items. As an alternative, a wide variety of structured P2P networks such as distributed hash tables (DHTs) have been recently proposed. Structured networks can efficiently locate rare items, but they incur significantly higher overheads than unstructured P2P networks for popular files. Through extensive measurements of the Gnutella network from multiple vantage points, we argue for a hybrid search solution, where structured search techniques are used to index and locate rare items, and flooding techniques are used for locating highly replicated content. To illustrate, we present experimental results of a prototype implementation that runs at multiple sites on PlanetLab and participates live on the Gnutella network.

1 Introduction

Unstructured networks such as Gnutella [1] and Kazaa [4] have been widely used in file-sharing applications. These networks are organized in an ad-hoc fashion and queries are flooded in the network for a bounded number of hops (TTL). While these networks are effective for locating highly replicated items, they are less so for rare items[1].

As an alternative, there have been proposals for using inverted indexes on distributed hash tables (DHTs) [8]. In the absence of network failures, DHTs guarantee perfect recall, and are able to locate matches within a small number of hops (usually $log(n)$ hops, where n is the number of nodes). However, DHTs may incur significant bandwidth for publishing the content, and for executing more complicated search queries such as multiple-attribute queries. Despite significant research efforts to address the limitations of both flooding and DHT search techniques, there is still no consensus on the best P2P design for searching,

In this paper, we measure the traffic characteristics of the Gnutella network from multiple vantage points located on PlanetLab [6]. Our findings confirm that while Gnutella is effective for locating highly replicated items, it is less suited for locating rare items. In particular, queries for rare items have a low recall rate (i.e., the queries fail to return files even though the files are actually stored in the network). In addition, these queries have poor response times. While these observations have been made before, to the best of our knowledge, our study is the first to quantify them in a real network. For example, we show that as many as 18% of all queries return no results, despite the fact that for two thirds of these queries, there are results available in the network.

We use extensive measurements to analyze the traffic characteristics of Gnutella, and based on our observations, we propose a simple hybrid design that aims to combine the best of both worlds: use flooding techniques for locating popular items, and structured (DHT) search techniques for locating rare items.

We find that such a design is particularly appropriate for existing P2P file-sharing systems in which the number of replicas follow a long tailed distributions: flooding-based techniques work best for the files at the head of the distribution, while DHT techniques work best for the files at the tail of the distribution.

To evaluate our proposal, we present experimental results of a hybrid file-sharing implementation that combines Gnutella with PIER, a DHT-based relational query engine [11]. Our prototype runs at multiple sites on the PlanetLab testbed, and participates live on the Gnutella network.

[1] In this paper, we will use the terms "files" and "items" interchangeably

2 Setting and Methodology

To analyze the Gnutella network, we have instrumented the Limewire client software [5]. Our client can participate in the Gnutella network either as an ultrapeer or leaf node, and can log all incoming and outgoing Gnutella messages. In addition, our client has the ability to inject queries into the network and gather the incoming results.

The current Gnutella network uses several optimizations to improve the performance over the original flat flooding design. Some of the most notable optimizations include the use of *ultrapeers* [3] and *dynamic querying* [2] techniques. Ultrapeers perform query processing on the behalf of their *leaf nodes*. When a node joins the network as a leaf, it selects a number of ultrapeers, and then it publishes its file list to those ultrapeers.

A query for a leaf node is sent to an ultrapeer which floods the query to its ultrapeer neighbors up to a limited number of hops. Our crawl reveals that most ultrapeers today support either 30 or 75 leaf nodes[2]. Dynamic querying is a search technique whereby queries that return fewer results are re-flooded deeper into the network. Our modified client supports both of these optimizations.

2.1 Gnutella Search Quality

To estimate the size of the Gnutella network, we began our study by performing a crawl of Gnutella. To increase the accuracy of our estimation, the crawl was performed in *parallel* from 30 ultrapeers for about 45 minutes. This parallel crawl was carried out on 11 Oct 2003 at around noon (Pacific time). The network size of Gnutella at the time of the crawl was around 100,000 nodes, and there were roughly 20 million files in the system.

Next, we turn our attention to analyzing the search quality of Gnutella, both in terms of *recall* and *response time*. The recall of a query is defined as the number of results returned divided by the number of results actually available in the network. Results are distinguished by filename, host, and filesize. Thus, each replica of a file is counted as a distinct result. Given the difficulty of taking a snapshot of all files in the network at the time the query is issued, we approximate the total number of results available in the system by issuing the same query simultaneously from all 30 PlanetLab ultrapeers, and taking the union of the results. This approximation is appropriate for the following two reasons. First, as the number of PlanetLab ultrapeers exceeds 15, there is little increase in the total number of results (see Figure 3). This suggests that the number of results returned by all 30 ultrapeers is a reasonable approximation of the total number of results available in the network. Second, because this approximation underestimates the number of total results in the network, the recall value that we compute is an overestimation of the actual value.

[2]This is confirmed by the development history of the Limewire software: newer Limewire ultrapeers support 30 leaf nodes and maintain 32 ultrapeer neighbors, while the older ultrapeers support 75 leaf nodes and 6 ultrapeer neighbors. As a side note, in newer versions of the Limewire client, leaf nodes publish Bloom filters of the keywords in their files to ultrapeers [7, 2]. There have also been proposals to cache these Bloom filters at neighboring nodes. Bloom filters reduce publishing and searching costs in Gnutella, but preclude substring and wildcard searching (which are similarly unsupported in DHT-based search schemes.)

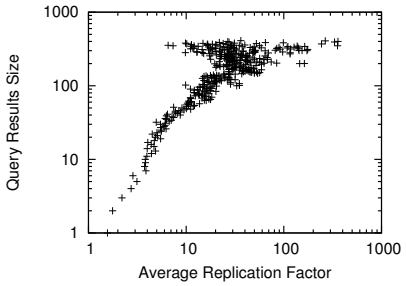

Figure 1: *Correlating Query Results Size vs. Average Replication Factor.*

We obtained Gnutella query traces, and chose 700 distinct queries from these traces to replay at each of the PlanetLab ultrapeers. To factor out the effects of workload fluctuations, we replayed queries at three different times. In total, we generated $63,000$ queries ($700 \times 30 \times 3$). We make three observations based on the results returned by these queries.

First, as expected, there is a strong correlation between the number of results returned for a given query, and the number of replicas in the network for each item in the query result set. The *replication factor* of an item is defined as the total number of identical copies of the item in the network. Again, to approximate this number, we count the number of items with the same filename in the union of the query results obtained by the 30 ultrapeers for the same query. We then compute the average replication factor of a query by averaging the replication factors across all distinct filenames in the query result set. Figure 1 summarizes our results, where the Y-axis shows query results set size, and the X-axis shows the average replication factor averaged across all queries for each results set size. In general, queries with small result sets return mostly rare items, while queries with large result sets return both rare and popular items, with the bias to-

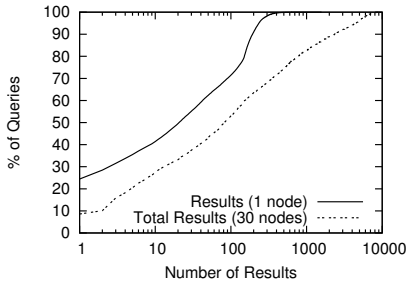

Figure 2: *Result size CDF of Queries issued from 30 Ultrapeers.*

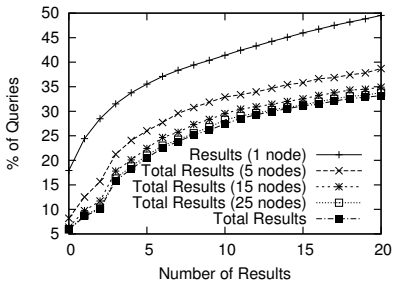

Figure 3: *Result size CDF for Queries* ≤ 20 *results.*

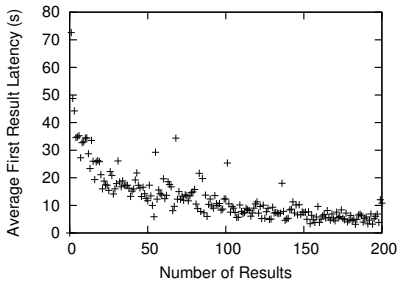

Figure 4: *Correlating Result Size vs. First Result Latency.*

wards popular items.

Second, our results demonstrate the effectiveness of Gnutella in finding highly replicated content. Figure 2 plots the CDF of the number of results returned by all queries (the *Results* curve), and a lower bound on the total number of matching items per query (the *Total Results* curve). We compute this lower bound by taking the union of all result sets obtained by the 30 ultrapeers for each query. Note that there are queries returning as many as 1,500 results, which would seem more than sufficient for most file-sharing uses. In addition, Figure 4 shows that the queries with large result sets also have good response times. For queries that return more than 150 results, we obtain the first result in 6 seconds on average.

Third, our results show the *ineffectiveness* of Gnutella in locating rare items. Figure 4 shows that the average response time of queries that return few results is poor. For queries that return a single result, 73 seconds elapsed on average before receiving the first result.

An important point to note is that queries that return few items are quite prevalent. Figure 3 shows the results of the same experiment as Figure 2, limited to queries that return at most 20 results for 5, 15 and 25 ultrapeers. Note that while 29% of queries receive more than 100 results, and 9% receive more than 200 results, there are 41% of queries that receive 10 or fewer results, and 18% of queries that receive *no* results. For a large fraction of queries that receive no results, matching results are in fact available in the network at the time of the query. Taking the union of results from all 30 ultrapeers for each query, the results improve considerably: only 27% of queries receive 10 or fewer results, and only 6% of queries receive no results. This means that there is an opportunity to reduce the percentage of queries that receive no results from 18% to at least 6%, or equivalently to reduce the number of queries that receive no results by at least 66%. We say "at least" because the union of results is an underestimation of the total number of results available in the network.

2.2 Increase the Search Horizon?

An obvious technique to locate more rare items in Gnutella would be to increase the search horizon using larger TTLs. While this would not help search latency, it could improve query recall. As the search horizon increases, the number of query messages sent will increase almost exponentially. Given that queries that return few results are fairly common, such aggressive flooding to locate rare items is unlikely to scale. In future work, we plan to quantify the impact of increasing the search horizon on the overall system load.

2.3 Summary

Our Gnutella measurements reveal the following findings:

- Gnutella is highly effective for locating popular items. Not only are these items retrieved in large quantities, the queries also have good response times.
- Gnutella is less effective for locating rare items: 41% of all queries receive 10 or fewer results, and 18% of queries receive *no* results. Furthermore, the results have poor response times. For queries that return a single result, the first result arrives after 73 seconds on average. For queries that return 10 or fewer results, 50 seconds elapsed on average before receiving the first result.
- There is a significant opportunity to increase the query recall for locating rare items. For instance, the number of queries that return no results can be reduced from 18% to at least 6%.

Thus, there are a considerable number of queries for rare items, and there is a considerable opportunity to improve the recall and response times of these queries. Furthermore, we note that flooding more aggressively is not an

answer to this problem, as flooding with a higher TTL will not necessarily decrease the response time, and will significantly increase the system load.

3 The Case for Hybrid

Various research efforts have proposed DHTs as an alternative to unstructured networks like Gnutella, arguing that DHTs can improve query performance. In this section, we explore the feasibility of a DHT-based query system. In a flooding scheme, *queries are moved towards the data*. In contrast, DHT-based search schemes move *both queries and data*, causing them to rendezvous in the network. This movement typically consists of two phases. First, a *content publishing* phase moves copies of data into traditional inverted files which are then indexed by keyword in a DHTs. They are also known as inverted indexes. Each inverted file comprises a set of unique file identifiers (a *posting list*) for a distinct keyword. Secondly, a *query execution* phase performs boolean search by routing the query via the DHT to all sites that host a keyword in the query, and executing a distributed join of the postings list entries of matching items.

While DHT-based search provides perfect recall in the absence of network failures, a full-fledged DHT implementation has its own drawbacks. The content publishing phase can consume large amounts of bandwidth compared to queries that retrieve sufficient results via flooding in an unstructured network. Consider the query "Britney Spears" that requests all songs from this popular artist. "Britney" and "Spears" are popular keywords with large posting lists. The publishing costs of building the inverted indexes for these two keywords are high. A "Britney Spears" query also requires shipping large posting lists to perform the distributed join. Recent back-of-the-envelope calculations [12] suggest that shipping large posting lists over DHTs is bandwidth-expensive. While compression techniques and Bloom filters would reduce the bandwidth requirements of publishing, a flooding scheme that does not incur any publishing overheads is both simpler and more efficient for such queries.

On the other hand, queries over rare items are less bandwidth-intensive to compute, since fewer posting list entries are involved. To validate the latter claim, we replayed 70,000 Gnutella queries over a sample of 700,000 files[3] using a distributed join algorithm over DHTs [11]. We observed that on average, queries that return 10 or fewer results require shipping 7 times fewer posting list entries compared to the average across all queries. This motivates a *hybrid search infrastructure*, where the DHT is used to locate rare items, and flooding techniques are used for searching highly replicated items.

[3]These queries and files were collected from 30 ultrapeers as described in Section 2.1.

3.1 Hybrid Techniques

The hybrid search infrastructure utilizes *selective publishing* techniques that identify and publish only rare items into the DHT. Different heuristics can be used to identify which items are rare. One simple heuristic is based on our initial observation in Section 2.1: rare files are those that are seen in small result sets. In essence, the DHT is used to cache elements of small result sets. This scheme is simple, but suffers from the fact that many rare items may not have been previously queried and found, and hence will not be published via a caching scheme. For these items, other techniques must be used to determine that they are rare. For example, publishing could be based on well-known term frequencies, and/or by maintaining and possibly gossiping historical summary statistics on file replicas.

This hybrid infrastructure can easily be implemented if all the ultrapeers are organized into the DHT overlay. Each ultrapeer is responsible for identifying and publishing rare files from its leaf nodes. Search is first performed via conventional flooding techniques of the overlay neighbors. If not enough results are returned within a predefined time, the query is reissued as a DHT query.

3.2 Network Churn

A practical concern of using DHTs is the network churn. A high network churn rate would increase the DHT maintenance overhead to manage publishing (and unpublishing). To understand the impact of churn, we measure the connection lifetimes of ultrapeer and leaf neighbors from two leaf nodes and two ultrapeers over 72 hours. The connection lifetimes we measure are a lower bound on the session lifetime as nodes may change their neighbor sets during the course of their Gnutella session. We make the following two observations.

First, the measured average connection lifetimes of leaf and ultrapeer nodes are 58 minutes and 93 minutes respectively. Ultrapeers have 1.5 times longer lifetimes than leaf nodes. To reduce the overheads of DHT maintenance, only stable ultrapeers with more resources should be used as DHT nodes.

Second, the measured *median* connection lifetimes of leaf and ultrapeer nodes are only 13 minutes and 16 minutes respectively. Since the median lifetime is much lower

than the mean, by discounting the short-lived nodes we have a fairly stable network. For instance, if we eliminate all leaf nodes whose lifetimes exceed 10 minutes, the average lifetime of the remaining nodes is 106 minutes[4]. In general, the longer a node is up, the longer one can expect a node to stay up. Hence, to address the issue of stale data in the DHT, file information of short-lived nodes should simply not be indexed. These short-lived nodes are not useful sources of data anyway since they are likely to disconnect before others can download their content.

4 Preliminary Experimental Results

To evaluate our hybrid design, we deploy a number of *hybrid* clients on PlanetLab that participate on the Gnutella network as ultrapeers. In addition, these clients are plugged into a DHT-based search engine built on top of PIER [11], a P2P relational query engine over DHTs. Our deployment should be seen as a strawman; a fully-deployed hybrid infrastructure would require an upgrade of all existing clients.

In addition to the traditional distributed join algorithm discussed earlier for searching, the PIER search engine also utilizes *Join Indexes*, by storing the *full text* (i.e. the filename) redundantly with each posting list entry. The search query is hence sent only to a single node hosting any one of the search terms, and the remaining search terms are filtered locally. This technique incurs extra publishing overheads, which are prohibitive for text document search, but tolerable for indexing short filenames.

Each hybrid ultrapeer monitors query results from its regular Gnutella traffic. These query results are responses to queries forwarded by the ultrapeer. Query results that belong to queries with fewer than 20 results are then published into the DHT. The publishing rate is approximately one file per 2-3 seconds per node. Each published file and corresponding posting list entries incurs a bandwidth overhead of 3.5 KB per file. Join Indexes increase the publishing overhead to 4 KB per file. A large part of the bandwidth consumption is due to the overheads of Java serialization and self-describing tuples in PIER, both of which could in principle be eliminated.

We test the hybrid search technique in PlanetLab on leaf queries of the hybrid ultrapeers. Leaf queries that return no results within 30 seconds via Gnutella are re-queried using the PIER search engine. PIER returns the first result within 10-12 seconds, with and without Join Indexes respectively. While decreasing the timeout to invoke PIER would improve the aggregate latency, this would also increase the likelihood of issuing extra queries. As part of our future work, we plan to study the tradeoffs between the timeout and query workload. Note that the average latency for these queries to return their first result in Gnutella is 65 seconds (see Figure 4). Hence, the hybrid approach would reduce the latency by about 25 seconds.

In addition, the hybrid solution reduces the number of queries that receive no results in Gnutella by 18%. This reduction serves as a lower bound of the potential benefits of the hybrid system. The reason why this value is significantly lower than the potential 66% reduction in the number of queries that receive no results is two fold:

- Unlike Gnutella measurements reported in Section 2.1 where queries are proactively flooded from many ultrapeers, in our experiment, we consider only the files that are returned as results to previous queries. Thus, this scheme will not return the rare items that were not queried during our experiments. Employing simple optimizations in which peers publish proactively their list of rare items should considerably boost the benefits of our scheme.

- As the number of clients that implement our scheme increase, we expect the coverage to improve as well. The coverage would be even better in a full-fledged implementation in which each ultrapeer would be responsible for a set of leaf nodes from which they would identify and publish rare items.

Using Join Indexes, each query needs to be sent to only one node. The cost of each query is hence dominated by shipping the PIER query itself, which is approximately 850 B. The distributed join algorithm incurs a 20 KB overhead for each query. These results indicate that the benefits of reducing per-query bandwidth might outweigh the publishing overheads of storing the filename redundantly, which makes Join Indexes a more attractive option.

5 Related Studies

A recent study [9] has shown that most file downloads are for highly replicated items. One might think that their findings contradict our analysis in Section 2.1 that shows that queries for rare items are substantial. However, the two studies focus on different aspects of Gnutella's workload. First, we measure result set sizes of queries, while their study measures download requests. Downloads only reflect successful queries, in instances when users have identified matching items from the result set that satisfied

[4]This is consistent with the results reported by Limewire's measurements of 300 connections over several days[7].

their search queries. This approach excludes queries that failed to find matching rare items even when they exist somewhere in the network, or return too few results that are of relevance to the search query. Second, both studies correctly reflect different aspects of the Zipfian distributions. Their study shows the *head* of the Zipfian popularity distribution, and hence they measure the download requests based on the items that match the top 50 query requests seen. In contrast, our study focuses on the long *tail* of the distribution as well. While individual rare items in the tail may not be requested frequently, these queries represents a substantial fraction of the workload, and are worth optimizing.

A separate study [10] has shown that the popularity distribution of a file-sharing workload is flatter than what we would expect from a Zipfian distribution. The most popular items were found to be significantly less popular than a Zipfian distribution would predict. Our proposed hybrid infrastructure would still apply here, utilizing flooding-based schemes for items in the "flattened head" region, and DHTs for indexing and searching for items in the tail of the distribution.

6 Conclusion

In this paper, we have presented the case for a hybrid search infrastructure that utilizes flooding for popular items and the DHT for searching rare items. To support our case, we have performed live measurements of the Gnutella workload from different vantage points in the Internet. We found that a substantial fraction of queries returned very few or no results at all, despite the fact that the results were available in the network. Preliminary experimental results from deploying 50 ultrapeers on Gnutella showed that our hybrid scheme has the potential to improve the recall and response times when searching for rare items, while incurring low bandwidth overheads.

7 Acknowledgements

The authors would like to thank Shawn Jeffery, Karthik Lakshminarayanan, Sylvia Ratnasamy, Sean Rhea, Timothy Roscoe, Scott Shenker, Lakshminarayanan Subramanian and Shelley Zhuang for their insights and suggestions. We thank the anonymous reviewers for their comments. This research was funded by NSF grants IIS-0205647, IIS-0208588, IIS-0209108 and Intel Research Berkeley.

References

[1] Gnutella. `http://gnutella.wego.com`.

[2] Gnutella Proposals for Dynamic Querying. `http://www9.limewire.com/developer/dynamic_query.html`.

[3] Gnutella Ultrapeers. `http://rfc-gnutella.sourceforge.net/Proposals/Ultrapeer/Ultrapeers.htm`.

[4] Kazaa. `http://www.kazza.com`.

[5] Limewire.org. `http://www.limewire.org/`.

[6] PlanetLab. `http://www.planet-lab.org/`.

[7] Query Routing for the Gnutella Network. `http://www.limewire.com/developer/query_routing/keywordrouting.htm/`.

[8] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking Up Data in P2P Systems.

[9] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making Gnutella-like P2P Systems Scalable. In *Proceedings of ACM SIGCOMM 2003*.

[10] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. Measurement, Modeling and Analysis of a Peer-to-Peer File-Sharing Workload. In *Proceedings of the 19th ACM Symposium of Operating Systems Principles (SOSP-19)*, Bolton Landing, New York, October 2003.

[11] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *Proceedings of 19th International Conference on Very Large Databases (VLDB)*, Sep 2003.

[12] J. Li, B. T. Loo, J. Hellerstein, F. Kaashoek, D. Karger, and R. Morris. On the Feasibility of Peer-to-Peer Web Indexing and Search. In *IPTPS 2003*.