Towards Feasibility and Scalability of Text Search in Peer-to-Peer Systems

Akshay Lal, Varun Gupta, Khaled Harfoush, and Injong Rhee
Email: [alal, vgupta, harfoush, rhee]@cs.ncsu.edu

***Abstract*— In this paper, we introduce a search engine, Dgoogle, designed for large scale P2P systems. Dgoogle is purely text-based, does not organize documents based on pre-defined keywords or based on their semantics. It is simple to implement and can tolerate variations in the wording of text queries. Compared to existing proposals, such as *Inverted Indices*, Dgoogle does not stress network bandwidth and offers an order of magnitude of savings in storage overhead and in the response time to user queries. Furthermore, Dgoogle's performance is not affected by long queries or by processing popular query words. Simulation results validate the efficacy of our proposal.**

I. Introduction

Searching text documents is an important functionality whether documents are co-administered, as is the case in Google-like web search engines, or scattered across different administrative domains, as is the case in distributed P2P systems. The de-centralized nature of P2P systems provides more service capacity, robustness to failures, resilience to attacks and to censorship by authorities. The same traits that make de-centralized systems attractive complicate the text search functionality. The fact that a search can potentially span thousands of peers in different administrative domains, the omnipresent delay between peers, and the fuzziness in the wording of text queries contribute to the difficulty of the problem [16]. Popular P2P systems either opt to neglect the problem altogether and query documents by specifying their full identity (file names) [20], [25], [5] or *flood* query messages to arbitrary peers and assume text search functionality at these peers [2], [29]. Also, proposals have been set forth for organizing documents in P2P systems based on their semantics and a semantic search is carried rather than a full text search [24], [7], [27], [6]. The most promising attempts to introduce full text search in P2P systems were proposed in [21]. The key idea is to implement a distributed version of Google's [4] *inverted indices*, lists mapping keywords to documents. A text query is communicated to peers maintaining the inverted indices of the query words. These lists are then intersected to obtain references to documents that contain *all* the query words. The limitations of this approach are that (1) documents are organized based on pre-defined keywords and thus it is difficult to tolerate variations in the wording of text queries. (2) Long queries and queries for popular words may stress the network bandwidth due to the large number of exchanged inverted indices. (3) The number of references to a document in the system depends on the number of words in this document. Thus maintaining large files in the system requires maintaining a large number of references to these files, which is a heavy burden especially in P2P systems, which are inherently dynamic and characterized by high churn.

In this paper, we propose a new text search engine, Dgoogle, designed to overcome these limitations. Dgoogle is purely text-based, does not organize documents based on pre-defined keywords or based on their semantics. It is simple to implement and can tolerate variations in the wording of text queries. Dgoogle does not stress network bandwidth and offers an order of magnitude of savings in storage overhead and in the response time to user queries. Dgoogle's performance is not affected by long queries or by processing popular query words. Furthermore, a maintained document is referenced in only one location in the system, unless caching is used, which leads to better resilience to system churn.

Dgoogle design relies on the following ideas: (1) In order to tolerate variations in the wordings of text, Dgoogle hashes *trigrams*, contiguous *three* letters/symbols in any text, instead of hashing keywords, and encodes a text document or a query, independent of its length, as an m-bit string, which we also refer to as the *signature*. Each text document/query, with signature d, can thus be mapped into a unique location of a hash space. (2) A document, with a signature d, should be retrieved from the system in response to a query, with signature q, if and only if q *derives to* d, $q \Rightarrow d$. This condition is satisfied if all set bits in q are also set in d. In other words a document should be retrieved if it has all the trigrams that exist in the query. For example, $10110 \Rightarrow 11110$. This abstraction is quite generic and has been used in different contexts [11]. (3) A key component of Dgoogle is the P2P structure to which document signatures are mapped and organized such that peers managing relevant documents for a query, satisfying the "$\Rightarrow$" property, can be efficiently retrieved. Dgoogle thus targets a P2P structure, which emulates the "$\Rightarrow$" property, basically an m-dimensional *hypercube*. It is however not desirable to manage a high-degree structure, m is in the order of thousands, especially in P2P systems characterized by high churn. Dgoogle resolves this issue by relying on a *grid* structure of an arbitrary degree. As a proof of concept, we implemented and tested Dgoogle using realistic data of more that 100,000 web pages. We

report on the storage and communication resources consumed and the response time to user queries compared to the inverted indices approach. We also suggest possible enhancements and identify future research directions.

The rest of this paper is organized as follows: In Section II we survey relevant text search techniques in P2P systems. In Section III we explain the hash algorithm that we use to generate text signatures. In Section IV we motivate the use of the *grid* structure to support Dgoogle's text search functionality. In Section V we provide the details for publishing, querying and load balancing the Dgoogle system. In Section VI we provide experimental results using a large realistic data set obtained from the Search Engine Optimization Resources [22]. We finally conclude in Section VII.

II. Related Work

There has been many attempts at introducing full text search in P2P systems. In [8], the authors propose gossiping in unstructured P2P networks to share compressed information about maintained documents. Gossiping in general does not scale to large scale systems. Along the same lines, the authors in [12] try to locate the content by selecting the nodes to search based on some heuristics like past behavior, past number of results, past query response time, etc. These methods obtain good level of efficiency in searching P2P networks, but the heuristic-based selection of the nodes to which queries are sent may fail to find relevant documents. In [26], the authors propose top-k posting list joins, bloom filters, and caching as promising techniques to reduce search costs for multi-term queries. A study reported in [30] shows that this solution cannot scale to web-size document collections.

Along a different direction, a set of techniques relying on inverted indices to carry text search in P2P systems have been proposed [4], [3], [18], [10]. MINERVA maintains a global index with peer statistics in a structured P2P overlay to identify the list of peers which are most likely to answer the user query [3]. Also, In [18] Lu et al. proposed ALVIS which also relies on inverted indices where every node is responsible for a set of words and maintain lists containing all the nodes in the system which have documents containing those words. In [10], the authors propose the Keyword-Set Search System (KSS), which pre-computes and stores intersection results of inverted lists of popular query keywords in advance. The authors generate exhaustive term combinations, which leads to unrealistic storage requirements for the index. A study presented in [30] shows that the network bandwidth is expected to be a bottleneck in systems relying on inverted indices. Our proposed Dgoogle (1) encodes trigrams as opposed to words, (2) stores only one reference to each registered document, unless caching is used, and (3) relies on a regular grid structure. Thus compared to the Inverted Indices approach, Dgoogle (1) does not organize documents based on pre-defined keywords or based on their semantics. (2) It is simple to implement and can tolerate variations in the wording of text queries. (3) Dgoogle's performance is not affected by long queries or by processing popular query words. (4) Provides better resilience to system churn since a document is not referenced in as many nodes in the system as the number of words in the document. (5) Dgoogle's storage requirements is much more manageable than the Inverted Indices appriach. (6) Dgoogle does not stress network bandwidth since only query messages are communicated between nodes and long Inverted indices do not need to be exchanged. (7) The response time to user queries is also much less since references to published documents can be returned to users without waiting for Inverted Indices to be communicated and intersected.

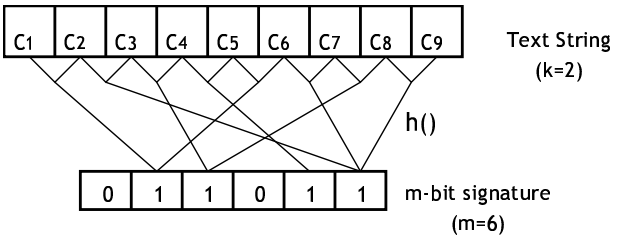

Fig. 1. Encoding a text string into its signature when $k = 2$ and $m = 6$.

III. Text Encoding

We use a variation of the technique suggested in [11] to compress any text, whether a document or a query, into m-bit string. This technique was suggested by Malcom C. Harrison in 1971 and later enhanced by Tharp et al. in [28].

The signature is obtained by hashing all contiguous k-symbol groups into an integer in the $[0...m-1]$ range. For each of these k-symbol groups, the bit position corresponding to the resulting hash value is set to 1. For example, consider the following text string: "*music classic*" where the *space* character is used as a delimiter between the two words. If we take $k = 3$, then all 3-symbol groups (*trigrams*) would be hashed; i.e. $h(mus)$, $h(usi)$, $h(sic)$, $h(cla)$, $h(las)$, $h(ass)$, $h(ssi)$, $h(sic)$ where $h()$ is the hash function and its range is the set of integers between 0 and $m-1$. Note that we do not include the space delimiter while hashing and hashing does not occur between characters from two different words; that is, we do not hash ccl (the last character of the *music* word and the first two characters of the *classic* word). This is done to take care of each word separately because words may appear in any order in a query or in a document. Furthermore, encoding k-symbol groups as opposed to full words takes care to a large degree of the fuzziness in wording of the text. Figure 1 shows an example of encoding a text string into its signature when $k = 2$ and $m = 6$.

The choice of the m and k parameters and the choice of the hash function $h()$ are critical to the success of

Dgoogle. Recall from Section I, that Dgoogle relies on a signature test, $\Rightarrow$. Clearly, the larger m is the more detailed the encoding and the more accurate the signature test, in the sense that fewer text strings will be incorrectly identified as satisfying the signature test. Also, if k is *one*, no information is included in the signature about the order, so we use $k \geq 2$.

The choice of $h()$ directly impacts Dgoogle's performance. In order to define the hash function $h()$, we place the symbols of the English language into classes so that the likelihood of occurrence is approximately the same for each group. This is because all symbols do not occur with the same frequency in the English language [15], [28]. Table I shows a sample classification of characters into $C = 10$ classes. Characters in the same class have the same hash value. For a substring $Y_w = y_0, y_1, ..., y_{w-1}$ of length w, then $h(Y_w) = \sum_{i=0}^{w-1} C^i T(y_i)$ where $T(y_i)$ is the class number corresponding to character y_i from Table I. Note that $0 \leq h(Y_w) < C^w$. Also, treating characters in the same class as equivalent, then there are C^w distinct substrings of length k and thus our choice of $m = C^k$. Note that the signature is not necessarily unique in the sense that the hashing technique can produce the same signature for two different input strings, which basically leads to *false positives*: strings signaled as satisfying the signature property, while they don't. This may happen only if the number of classes C is smaller than the number of symbols in the alphabet. When this is not the case, then there are no false positives. However, encoding k-symbol groups instead of words can lead to another problem: different words may have the same k-symbol groups. For example, a query for the word "bible" may lead to a document which has the trigrpahs "bib", "ibl", and "ble" but in words other than "bible". We label this source of error as *false order*. In Section VI we quantify the impact of false orders on the accuracy of Dgoogle's results .

Class	Characters
0	W E 1 ! + *
1	T 2 ? < @
2	A G 3 . > /
3	O Y 4 & (
4	I F Q 5 ; =)
5	N M J 6 : - {
6	S U K 7 _ # }
7	R C V X 8 ' ∧ ~ [
8	H B Z 9 %] .
9	D L P 0 " $ 1

TABLE I
CLASSIFICATION OF CHARACTERS

As a proof of concept, we consider only an alphabet of 37 characters *[A-B], [0-9], '.'*. To avoid false positives, we place each character in a separate class, $C = 37$, and consider trigrams, $k = 3$. Thus the number of bits in the signature is $m = 37^3 - 1 = 50652$, which is quite large and results in storage and communication inefficiency (5 packets of 1500 bytes). In order to reduce m into a manageable length, we note that many of 50652 bits correspond to trigrams that are not useful in the English language such as "xyz". In order to separate the wheat from the chaff, and prune bits corresponding to irrelevant trigrams we rely on the Aspell dictionary [1]. By parsing all dictionary words we identify relevant trigrams and only construct the final signature from bits corresponding to relevant trigrams. Odd trigrams appearing in our data set are mapped to one extra bit (*wildcard* bit). A simple table lookup allows us to implement this functionality and the resulting value of m reduces to 7804 bits corresponding to 7803 trigrams appearing in the dictionary and the wildcard bit. This means that almost 85% of all possible permutations of our alphabet were not useful for the english language.

IV. THE GRID STRUCTURE

In this section we target a design of a hash space in which m-bit signatures are mapped while allowing each signature x to easily *reach* other signatures y for which $x \Rightarrow y$. As peers are mapped to the same hash space and manage documents with contiguous signatures, this hash space arrangement allows a peer managing a document with signature x to easily connect to and communicate with peers managing documents with signature, y.

To simplify our discussion, consider 4-bit signatures, $m = 4$. In Figure 2 we plot the directed graph including all possible 4-bit signatures such that for any two signatures, x and y, there exists a path from x to y if and only if $x \Rightarrow y$. We label such graph the *transition diagram*. It is clear that it takes the shape of an m-dimensional hypercube, with each vertex having a degree of $m = 4$. In this hypercube a vertex representing signature $S = s_1 s_2 \dots s_m$ has fanout edges extended to other vertices representing signatures for which the same bits in S are also set in addition to one other set bit. In general, the *in-degree* at any signature is not equal to the *out-degree*. Representing the hash space as a hypercube seems plausible since it is a regular graph and most DHT-based P2P systems attempt to build regular overlay structures but this representation suffers two serious deficiencies.

First, while the hypercube is a regular graph, a *directed* hypercube as shown in Figure 2 (left) when used as the hash space in a DHT-based P2P system will *not* lead to a regular overlay structure. This is because peers only need to establish and maintain their *out* connections and as can be seen in the figure the out-degree differs for different signatures. This phenomenon is much more pronounced for larger values of m. Second, the fanout at most vertices will be large for large values of m. Recall from Section III that we use $m = 7804$. This value may be optimized but at the cost of introducing false positives and it is unlikely to be brought down to a very small number of bits. It is challenging for any peer to maintain a large number of connections to other peers given the high

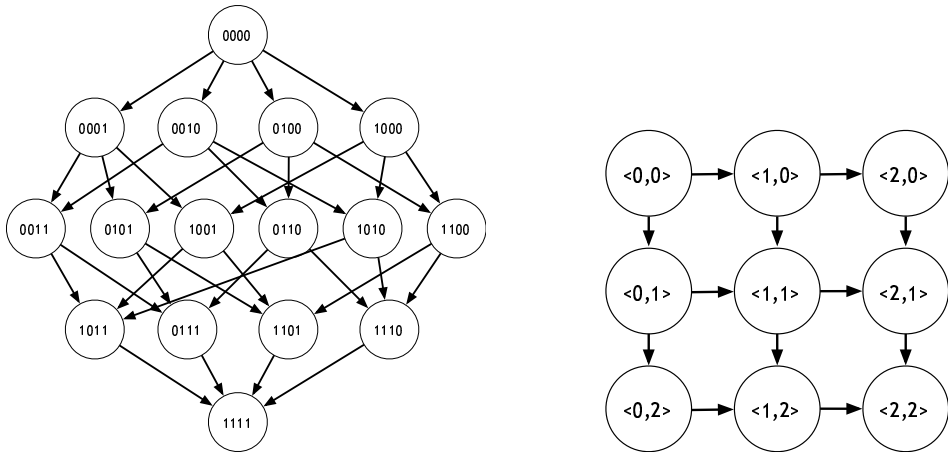

Fig. 2. A 4-dimensional cube corresponding to the transition diagram for 4-bit signatures (left). 2-dimensional grids corresponding to the transition diagram (right). Query messages start at the appropriate node in a diagram and then follow the direction of the arrows visiting potentially relevant nodes.

churn inherent in P2P systems. A hash space with regular structure and small degree is needed and can be achieved with a straightforward solution: By converting the m-dimensional hypercube structure into n-dimensional *grid* structure, where $n < m$ is a tunable parameter.

Consider an m-bit signature $S = s_1 s_2 \dots s_m$. The coordinate, $< C_1, C_2, \dots, C_n >$, of S in an n-dimensional *grid* would be obtained by splitting the m bits into n different portions, I_1, I_2, $\dots I_n$. If m is divisible by n, then $I_1 = s_1 \dots s_{m/n}$, $I_2 = s_{m/n+1} \dots 2s_{m/n}$, etc. If m is not divisible by n then the n-th portion may have a fewer number of bits than the others. Now a coordinate, C_i, is computed as the number of set bits in I_i (not the corresponding integer value). For example a signature $S = 1100$ would have a coordinate of $< 2, 0 >$ in a two-dimensional grid. Figure 2 (right) shows the two-dimensional grid representing the transition diagram for 4-bit signatures. In general, a vertex with coordinate $C =< C_1, C_2, \dots, C_N >$ has fanout edges extended to other vertices with coordinates matching C except in one dimension, which has a value larger by *one*. As a result, the vertex representing a signature $S = s_1 s_2 \dots s_m$ has fanout edges extended to other vertices representing signatures for which the same bits in S are also set in addition to one other bit set, which maintains the "$\Rightarrow$" sense of *proximity* and *direction* in the hash space.

Converting the hypercube to a grid results in a regular hash space structure but, as opposed to the hypercube case, each vertex may represent many signatures. Furthermore, the number of signatures per vertex can vary dramatically. Some vertices, especially towards the middle of the grid, handle many more signatures than vertices towards the edges of the grid. Thus this conversion has solved the irregularity in the number of connections at the cost of introducing irregularity in the number of signatures per vertex. However, a large number of signatures assigned to a vertex does not necessarily mean that the peer managing this vertex in the system will be overloaded. In fact, our results indicate that peers managing the vertices with most signatures are not necessarily the ones handling more documents. The imbalance in the number of files managed by different peers depends on the set of documents managed by the system and are better dealt with in runtime, as documents are registered in the system. We discuss load balancing in Section V.

V. DGOOGLE SEARCH ENGINE

The grid-like hash space advocated in Section IV forms the basis for the Dgoogle system. Peers and documents are hashed into this hash space as is generally done in any structured DHT-based P2P system such as [20], [25], [31], [17]. A new peer, p, joining the system contacts a bootstrap server to identify an entry point to the system and use it to reach the peer, q, managing the point in the hash space corresponding to its signature, then splits the hash space *area* that is managed by q, and manages the documents that fall into its area of the hash space. Unless otherwise specified, all basic functionality of the system such as handling node joins, departures, document publishing, querying, handling churn are handled as described in any DHT-based system. We next describe some basic text search functions in Dgoogle.

A. Publishing a Document

In order to publish a document in the system, a node encodes the document into an m-bit signature ($m = 7804$) as in Section III and obtains the document's coordinates in an n-dimensional grid as explained in Section IV. The signature and 128 byte abstract of the document are forwarded towards the peer managing the document's coordinates. The abstract and the signature are stored at that peer together with a link to the publishing node (IP address and port number). The signature size and the abstract size mount to 1104 bytes.

B. Querying the Network

A query is converted to an m-bit signature and its corresponding grid coordinate is computed before being forwarded to the peer managing this coordinate in the grid. The peer compares the signatures of all its managed documents to the query signature and returns to the client issuing the query (1) the abstracts of the documents with signatures satisfying the query signature, and (2) appropriate links to nodes that published these documents. The process does not stop there as matching results could also be found at other peers that can be reached by following the directed links of the grid.

A client issuing a query typically does not expect to receive all documents matching its query at the same time. Instead, the user is presented with only a handful of the most relevant documents. Further results require further action from the user such as clicking a button in his browser to extend the search whether from cached results on his machine or from the P2P system. This observation helps avoid the need to contact all peers that may contain answers to the query. The scope of the search can be easily controlled by pre-defining the search scope in the grid or by setting a TTL value in query messages. Either way, peers finding answers to the query will return them directly to the client and extending the search can be triggered by the client to uncover another relevant portion of the grid. Caching mechanisms can be used to reduce the response time to user queries.

C. Balancing the System Load

Balancing the load in distributed systems is a popular problem that has been addressed by many researchers. Examples include the efforts in [13], [9], [19], [14]. Any of the proposed techniques could be used to balance the load in Dgoogle. As a proof of concept, we rely on a simple load balancing approach. Recall from our discussion in Section IV that the number of signatures mapping to the grid vertices vary from vertex to the other. This does not necessarily mean that peers managing vertices corresponding to more signatures will have to do more in the system as it depends on the actual published documents and the popularity of these files. In order to avoid stressing a peer and also to optimize the response time to queries, we assume that there is a pool of peers willing to serve in the system and these nodes are invited to serve based on the system's need (as serving peers are stressed or to provide data redundancy, etc.). If a peer becomes overloaded, it calls on another peer to join and splits its hash space zone and its managed documents with the new peer. This is quite different from the traditional DHT-based P2P systems, in which all nodes typically join the system and at random locations in the hash space. As a proof of concept, we use the number of actual documents managed by a peer as a measure of its stress condition and do not allow a peer to manage more than MAXFILES documents. If the number of documents at a peer exceeds this limit, a new peer is invited to join the system. In Section VI we study the implications of the MAXFILES limit on performance.

VI. Experimental Results

To test the validity of our design and compare its performance to existing distributed text search techniques, we simulated the proposed system using a realistic data set of web documents. The simulations were carried on a single threaded AMD Athalon Xp 3000+ running Fedora Core 4. The code was written in C++ using the STL libraries for string comparison. We next provide more simulation details and results.

A. Data Set

The data set we used to populate our system consists of 100318 files, which were collected from the Internet using the popular unix utility *wget*. These files contained news websites such as CNN, BBC, TimesOfIndia, GoogleNews; technology reviews such as Slashdot, Cnet, Wired, ArsTechnica; sports news such as ESPN, TenSports, TheFa; business reviews such as TheBussinessTimes, Forbes; fashion magazines such as Elle, FTV; and an assortment of other random topics such as cooking, music, art etc. The largest collected file size being about 23.9 MB and the average being 32.3KB. In the signatures corresponding to the collected files, the largest number of bits set per signature is 4228 bits and the average is 1029 bits. Recall that our signature length, m, is 7804 bits.

The words used to query the system were taken from internet surveys presented by the Search Engine Optimization Resources [22]. This website has weekly and monthly surveys of the most popular queries searched for using Google, Yahoo, MSN and other major Internet search engines. We also randomly combine words to form multiple words queries. Specifically, we prepared a set 1000 one-word queries, a set 1000 two-word queries, and a set of 1000 three-word queries to test the system.

B. Performance Results

1) Encoding Accuracy: As mentioned in Section III, our encoding scheme can lead to *false orders*. We define the *encoding accuracy* as the fraction of queries not leading to false orders at all. Using the set of 100318 web pages, our results indicate that the encoding accuracy is 0.9 for one-word queries, 0.95 for two-word queries, and 0.986 for three-word queries. Given that the average query has roughly 2.5 words [21], it is clear that the pollution induced by false orders is reasonably small.

2) Storage Overhead: Dgoogle stores only one copy of a published document's information at a designated peer (the one managing the document's coordinate). This information includes the document's signature (7804 bits), abstract (128 bytes), and some additional metadata such as the IP address and port number of the publishing node, for a total of roughly 1105 bytes. The total amount of

storage needed would mount to this number multiplied by the number of published documents. Our load balancing technique proposed in Section V ensures that no peer will handle more than MAXFILES documents. If we set MAXFILES to 100 then in the worst case, a peer will store a little more than 100KB, and if we set MAXFILES to 1000 then in the worst case, a peer will store a little more than 1MB. These numbers are quite reasonable by today's standards and are definitely not prohibitive, even without further optimizing the size of the signatures. Compared to the inverted indices approach, which stores in the system as many references to a document as there are keywords in this document, the storage requirements in Dgoogle are less constraining.

3) Messaging Overhead: A query message is supposed to reach its designated peers and may be further propagated towards other peers. In [23] it has been noticed that 85% of all users only look at the first page of results put forth by Google.com and Yahoo.com, which by default is set to display 10 results. We thus define the messaging overhead as the cost (in bytes) of the messages exchanged between peers, starting from the time the query reaches the peer managing the query coordinates in the grid, until at least 10 results are returned to the user. A query message has a query ID to avoid processing the same query multiple times by the same peer. A message consists of the query signature, a query identifier, and the IP address and port number of the user issuing the query, and thus the size of a message is close to 1KB.

In Figure 3 we plot the average messaging overhead for one, two and three-word queries, as we vary the grid dimension and the MAXFILES of documents maintained by each peer. It is clear from the figures that (1) By increasing the grid dimension and/or the MAXFILEs limit, the messaging overhead can be reduced. This is expected since increasing the dimension helps better organize documents in the network and increasing the MAXFILES limit provides peers with more information. Both factors help reduce the need to visit more peers in order to retrieve at least 10 answers for a query. (2) The overall messaging overhead from each query is small especially for reasonably large dimension and MAXFILES (close to 13KB). (3) The messaging overhead for one-word queries is smaller than two and three-word queries. This can be explained by the fact that less peers need to be visited with one-word queries as these queries are more *general* and more results exist in the system for one-word queries than for two and three-word queries. The difference between two and three-word queries is negligible in our data set as most queries are answered within one hop from the peer managing the query coordinates. When compared to a system that is implemented using Inverted Indices, the Dgoogle messaging overhead is expected to be much less . The main reason being that Dgoogle does not require an exchange of as many lists as there are query words and carrying an intersection of these lists before identifying the query results; instead, the results are readily available at visited network peers. Note that the size of query messages can be drastically reduced. As a query typically has two or three words, instead of including 7804 bits in the messages, the location of the set bits in the signature could replace the signature, which is expected to bring the messaging overhead down by an order of magnitude. Furthermore, in the Inverted Indices approach, the number of propagated lists depends on the number of query words. While most most queries are upto three words, about 30% of the queries have more than three words [21]. In Dgoogle, the number of query words has minor impact on the messaging overhead if any. It should be clear that if caching is utilized then the messaging overhead will be reduced further and so does the response time, which we discuss next.

4) Response Time: The response time to user queries is the most important metric from a user's perspective. To simplify our measurements, we assume that the communication delay is the same between all peers and ignore the time to reach the query's designated peer and the time to transmit back the results to the user as these are inherent costs in any P2P system even those without text search capabilities. We thus measure the response *time* in hops and not in time, and define it as the maximum number of successive hops traversed by a query message. In Figure 4 we plot the average response time (in hops) for one, two, and three-word queries as we vary the grid dimension and MAFILES limit. The figures show that (1) the large dimension and MAXFILES values help reduce the response time; (2) for large enough values of the grid dimension and MAXFILES, results can reached in almost one hop; (3) the sensitivity of the response time to the query size is not pronounced. It should be noted that these results do not include queries for which 10 results do not exist in the system. In such a case, the query message will follow the grid edges all the way to the last coordinate (all 1's) without finding answers. Such problem can be curtailed by assigning a TTL value to query messages. The results in Figure 4 show that, for reasonably large dimension and MAXFILES values, a TTL of two hops is appropriate. Compared to the Inverted Indices approach, the user is expected to see results much faster and quite possibly starting as soon as the peer managing the query coordinates is reached without waiting for an exchange of the inverted indices and carrying the intersection. If communication with neighboring peers is needed, it is done in parallel and the results are returned to the user by these peers also in parallel.

VII. Conclusions and Future Work

We have introduced a new system capable of handling full text queries in distributed P2P systems. The text encoding can tolerate to some degree fuzziness in the wording of text queries. The system is not demanding whether in storage or in communication resources and

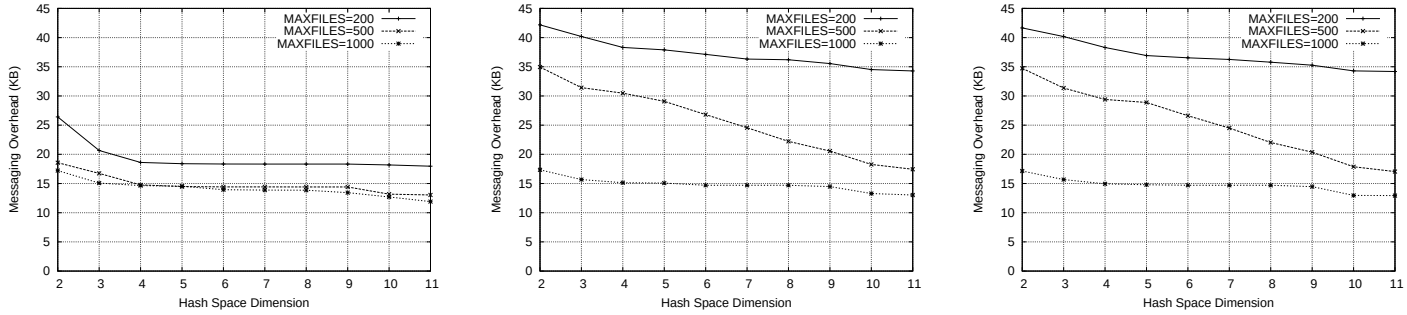

Fig. 3. Average messaging overhead for one-word queries (left), two-word queries (middle), and three-word queries (right).

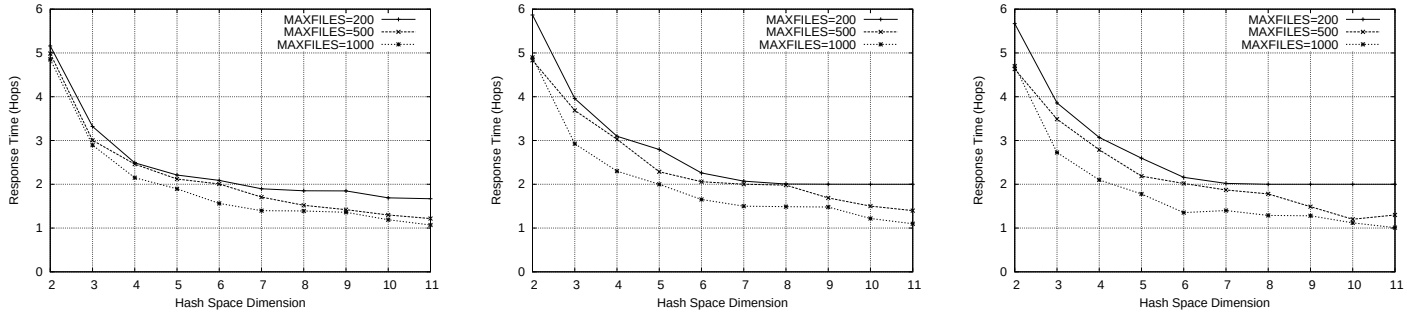

Fig. 4. Average Response Time (in Hops) for one-word queries (left), two-word queries (middle), and three-word queries (right).

offers fast response time to user queries. Still, the proposed system is a small step forward. There are many issues that need to be addressed and many enhancements to be considered. For example, ranking documents in a distributed setup is not trivial. Google relies on the popular *page ranking* algorithm to rank documents [4]. Other ranking metrics have also been proposed [21]. One approach to accommodate rankings into the Dgoogle system is to manipulate multiple grids each is responsible for managing documents of ranks within some range. A document is maintained in one of these grids depending on its rank and documents may be upgraded or downgraded to other grids if their importance ranking changes. Furthermore, the system needs to be resilient to peers trying to maliciously impact their documents' rankings. It would also be useful to equip the system with the ability to suggest alternative keywords in response to possible misspellings in the query words. We intend to further investigate these points in our future work.

REFERENCES

[1] http://aspell.sourceforge.net/.

[2] http://www.gnutella.com.

[3] M. Bender, S. Michel, P. Triantafillou, G. Weikum, and C. Zimmer. Improving collection selection with overlap awareness in p2p search engines. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 67–74, New York, NY, USA, 2005. ACM Press.

[4] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[5] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46–??, 2001.

[6] E. Cohen, A. Fiat, and H. Kaplan. Associative search in peer to peer networks: Harnessing latent semantics, March 2003.

[7] A. Crespo and H. Garcia-Molina. Semantic overlay networks for p2p systems, 2002.

[8] F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. In *Twelfth IEEE International Symposium on High Performance Distributed Computing (HPDC-12)*. IEEE Press, June 2003.

[9] G. Giakkoupis and V. Hadzilacos. A scheme for load balancing in heterogenous distributed hash tables. In *PODC '05: Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 302–311, New York, NY, USA, 2005. ACM Press.

[10] O. Gnawali. A keyword set search system for peer-to-peer networks, June 2002. Master's thesis, Massachusetts Institute of Technology.

[11] M. Harrison. Implementation of the substring test by hashing. In *Communications of the Association for Computing Machinery 14(12)*, 1971.

[12] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti. A local search mechanism for peer-to-peer networks. In *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management*, pages 300–307, New York, NY, USA, 2002. ACM Press.

[13] D. R. Karger and M. Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *SPAA '04: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 36–43, New York, NY, USA, 2004. ACM Press.

[14] K. Kenthapadi and G. S. Manku. Decentralized algorithms using both local and random probes for p2p load balancing. In *SPAA '05: Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 135–144, New York, NY, USA, 2005. ACM Press.

[15] D. Knuth. *The Art of Computer Programming, Volume 1, Fundamental Algorithms*. Addison-Wesley, Cambridge, MA, 1973.
[16] J. LI, B. LOO, J. HELLERSTEIN, F. KAASHOEK, D. KARGER, and R. MORRIS. the feasibility of peer-to-peer web indexing and search, 2003.
[17] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh. Graph-theoretic analysis of structured peer-to-peer systems: Routing distances and fault resilience. In *Proceedings of ACM SIGCOMM '03*, August 2003.
[18] T. Luu, F. Klemm, I. Podnar, M. Rajman, and K. Aberer. ALVIS Peers: A Scalable Full-text Peer-to-Peer Retrieval Engine. In *Workshop on Information Retrieval in Peer-to-Peer Networks P2P-IR at CIKM 2006*, 2006.
[19] A. Mondal, K. Goda, and M. Kitsuregawa. Effective loadbalancing of peer-to-peer systems, 2003.
[20] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *SIGCOMM '01*, San Diego, CA, August 2001.
[21] P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *Proceedings of International Middleware Conference*, pages 21–40, June 2003.
[22] SEOpen Search Engine Optimization Resources. http://seopen.com/.
[23] C. Silverstein, M. Henzinger, H. Marais, and M. Moricz. Analysis of a very large altavista query log. Technical Report 1998-014, Digital SRC, 1998. http://gatekeeper.dec.com/pub/DEC/SRC/technical-notes/abstracts/src-tn-1998-014.html.
[24] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient content location using interest-based locality in peer-topeer systems, 2003.
[25] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. balarkrishnan. Chord: A Peer-To-Peer Lookup Service for Internet Applications. In *SIGCOMM '01*, San Diego, CA, August 2001.
[26] T. Suel, C. Mathur, J. wen Wu, J. Zhang, A. Delis, M. Kharrazi, X. Long, and K. Shanmugasundaram. Odissea: A peer-to-peer architecture for scalable web search and information retrieval. In *WebDB*, pages 67–72, 2003.
[27] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks, August 2003.
[28] A. Tharp. *File Organization and Processing*. John Wiley & Sons, 1988.
[29] B. Yang and H. Garcia-Molina. Efficient search in peer-to-peer networks. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS)*, 2002.
[30] J. Zhang and T. Suel. Efficient query evaluation on large textual collections in a peer-to-peer environment. In *P2P '05: Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing (P2P'05)*, pages 225–233, Washington, DC, USA, 2005. IEEE Computer Society.
[31] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California at Berkeley, 2001.