CHORA: Expert-based P2P Web Search

Halldor Isak Gylfason[1], Omar Khan[1], and Grant Schoenebeck[1]

Electrical Engineering and Computer Science
University of California, Berkeley
{halldor, omar, grant}@cs.berkeley.edu

Abstract. We present CHORA, a P2P web search engine which complements, not replaces, traditional web search by using peers' web viewing history to recommend useful web sites to queriers. CHORA is designed around a two-step paradigm. First, CHORA determines which peers to query and then it executes a query across these peers. Each peer uses a desktop search engine to query their local web history and retrieve results ordered by relevance. To determine which peers to query, a small sketch of the information available from each peer is stored in a DHT. Peers with sketches indicating that they may have relevant information are queried. The query is dispersed through an ad hoc network connecting only those machines in the query and is optimized for getting good results as quickly as possible.

1 Introduction

While P2P Web search is not a new technique, most of the previous work in this area focuses on creating a P2P network which searches the *entire* Web [1–4]. Previous P2P Web search efforts have grown out of two observations: (1) even the largest centralized search engines probably cannot index the entire web and (2) Distributed Hash Tables (DHTs) [5, 6] provide an efficient way to index large amounts of data in a distributed manner. However, there are problems with these observations and the systems built upon them. One simple proof that there is something wrong is that these systems, some of which have been deployed commercially, have not been very successful.

We believe that one crucial observation is missing: you cannot replace a web search engine solely with promises of the future. With all P2P systems you need a bootstrapping mechanism: a way to grow the system until it reaches a sufficient number of users to sustain itself. None of the previous systems describe an effective bootstrapping mechanism. The added benefits that these systems claim, such as resistance to censorship and ability to search over a larger database, exist only in the future, and until these systems have many users, they are essentially an incomplete reproduction of a centrally hosted search engine.

By recognizing the extra observation above, we designed CHORA not to replace centralized Web search engines, but rather to use P2P search in the areas of Web search where it can offer the most improvement. The design of CHORA was motivated by a few scenarios where additional peer information could potentially improve search:

1. You find a new band that you like, say, *Arcade Fire*. You want to find out more about this band: who else likes their music and what they enjoy about it?
2. You need to concatenate two `pdf` files but do not know how.

In the first case, there is a group of people, *experts*, who will be able to answer the query well. If you could find and directly contact these experts for advice, they would be able to show you the results they found most useful. In the second case, you are seeking the same item which a large group of people has already sought and found. If you can identify these pages, you could use their experience to expedite your own search.

P2P search is a natural way to aggregate the experiences of individual users. P2P search interacts directly with user nodes, which contain a user's full browsing experience, along with many other facets of their computing history. On the other hand, web search engines, for the most part, only have access to result clicks, and suppositions inferred from those clicks.

CHORA, facilitated by using a two-phase query, provides a means of aggregating a user's rich computing history. In the first phase, *sketch query*, it queries a database stored in a Distributed Hash Table (DHT) to obtain a list of peers which are most likely to have relevant information. This DHT contains a sketch of what is on each computer, each sketch being a collection of the most important keywords for each user. From this query CHORA builds a *target list* of peers to query. During the second phase, *peer query*, CHORA delivers a query to each of these peers and aggregates the results.

Our two-phase execution works by limiting the number of peers queried by first identifying who is likely to be able to give relevant information. This necessity is born out of two limitations: it is infeasible to query every peer as has been seen in P2P applications such as Gnutella [7–9]. On the other hand, users are weary about the privacy of their web-browsing history. With a full index, in a centralized server or on a DHT, this information has to leave the user's computer wholesale. In CHORA the only information that needs to leave the peer's computer is a small sketch, which the user can filter.

There are two major contributions of this work: the observation that much of a single user's browsing history (along with all their computer interactions) is readily available to help supplement the web search results of *all users*; and a method that generates a sufficient description of the content of each user's computer, in order to efficiently direct queries to users likely to have good results. For the method presented in this paper, we use a user's browsing history and behavior as available in desktop search products like Google Desktop [10] to concisely capture their expertise. The main abstraction used is the concept of a click graph, which helps organize a user's web pages based on connectivity implied by their clicks and summary statistics describing their interaction with each page.

Previous work has not fully explored the method of first locating peers using sketches. Many have thought that it would simply be too difficult to find information stored in this way. A contribution of this paper is a study on how feasible it is. An additional contribution is the design and implementation of CHORA which efficiently routes queries, in order to receive good results as quickly as possible.

The paper is organized as follows: Section 2 explains the systems related details of the implementation. Section 3 explains how CHORA generates sketches and aggregates query results, while Section 4 contains an evaluation of CHORA. Finally, Section 5 briefly surveys relevant previous work and Section 6 concludes.

2 System Architecture

In this section we describe the architecture of CHORA. Our implementation uses the Python programming language, the OpenDHT Distributed Hash table [11] and Google Desktop.

2.1 Query Setup

Initially, when a user joins CHORA, the user's set of keywords is computed, as described in Section 3, and subsequently each keyword is registered in OpenDHT under the SHA1 hash of the keyword. Each registered keyword contains meta-data such as the URL of the computer, location/bandwidth parameters, and a set of related keywords on that computer. This is done to optimize queries involving multiple query terms, where we wish to find computers containing all of those keywords. Finally, every computer registers under a common fixed key, so the system can optionally flood queries to all participants – an option clearly feasible in the first stages of deployment.

2.2 Phase 1 - Sketch Query

When a user issues a query to CHORA, the system attempts to locate computers that are likely to have good results, based on each machine's list of keywords. This list of peers is referred to as the *target list*. Conceptually, a *Get* is performed to OpenDHT on each word in the query, and the peers whose sketches contain the most keywords are computed and prioritized. This can be optimized by using the meta-data that stores a set of other keywords for that computer, or by a mechanism embedded in the DHT such as PIER's in-network joins [12]. Once a target list has been constructed, the system moves to the peer query phase.

2.3 Phase 2 - Peer Query

At this stage, the query needs to be distributed to all the nodes in the target list. Since in the OpenDHT model, the nodes are not participants in the DHT routing, they must build a routing mechanism external to the DHT. In this model each node has greater control over the routing than it would have if the routing were embedded in a multicast enabled DHT. In particular, queries can first be sent to the nodes most likely to have good results.

For a large P2P network, it is likely infeasible for the issuing node to send the query directly to all of the related peers, so a query dissemination tree is constructed with the issuing node as the root. Each node in the tree receives the query from its parent, along with a list of computers which will become that node's descendants. The node then selects some of the descendant nodes from that list to become its children, divides the rest of the list among them, and forwards the query. Then the node executes the query locally using Google Desktop, and sends the local query results upstream. For fault tolerance, a list of ancestors can be maintained, so single node failures will not break the tree.

When constructing the tree it is important to be aware of the location of the nodes. Clearly, when selecting children it is important that the latency is as low as possible; a node in North America should not distribute its query to China, when it can choose other nearby nodes. Furthermore, nearby nodes are not equal. Some have access to more bandwidth, are more likely to have relevant information, have longer up-times and are more reliable. This observation has lead to a super-node architecture (e.g. Gnutella ultrapeers). The super-node architecture contains two types of nodes: *leaf-nodes* and *super-nodes*. The main tree is composed solely of super-nodes and the leaf-nodes are leaves of a super-node in the tree. In particular, only super-nodes have children. One advantage of this model is that slow nodes can be placed near the root of the tree without negatively affecting the time it takes results from other nodes to return.

Detecting near-by nodes can be done in various ways. In CHORA, query trees are short lived so we do not want to do any costly probing of individual machines at the time of the query. Thus, we opted for a network coordinate system to compute the latency [13]. Every node stores its Internet coordinates in the keyword meta-data. The state in the DHT is soft, so this registration is continuously updated to reflect changes in the coordinates.

We implemented 3 different methods of constructing the query tree:

1. **Random:** Here each node selects its children randomly, while respecting some system-configured fan out.
2. **Location based:** In this model each node selects the children to be the closest nodes according to the network coordinates, again respecting some system-configured fan out.
3. **Location based with super-nodes:** In the super-node model each node selects two types of children: super-children and leaf-children. Thus if a super-node is sufficiently close it will be selected as a super-child, despite the fact that some other nodes may be slightly closer. The primary goal when selecting leaf-children is to select nodes with data relevant to the query, in order for the user to more quickly receive good results. Each node decides for itself if it is a super-node, and may include metrics such as bandwidth, uptime or be statically configured. This indicator is stored in the DHT meta-data.

We have simulated these three approaches and results indicate that with the super-node approach we are successful in getting more results in early, and since the tree is constructed with nodes near the root that are likely to have good results, the user will notice considerable improvement.

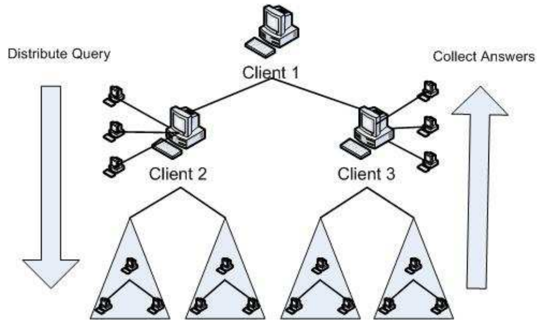

Fig. 1. Query Execution with super-nodes

3 Search Quality

3.1 Sketches

Chora's two-step query relies on good sketches in order to locate peers who have information relevant to the query. Because the DHT can only look up keywords, we are restricted to describing users by a list of searchable keywords (assuming a user population large enough to render flooding impractical).

Recall the goal behind these sketches: each user wants to advertise topics for which they have already done extensive research so that other users can leverage that research. For example, Chora can leverage users that might have no long-term interest in a topic like "DVD players," but who are marked as an expert on "DVD players" because they recently performed extensive research on DVD players in preparation for a purchase. We observe that the users who share similar interests are only good users to query if the query is related to their shared interests. Therefore, characterizations of peers that are not query sensitive (i.e. bookmarks) are insufficient for our purpose.

Because these sketches are stored on another system, privacy is a concern. Mechanisms can be added to allow the user to filter the keywords in sketches published by his computer or to disallow certain queries from being accepted by his computer. Another way to ensure privacy is to obfuscate the results from each peer. For example, each peer could be made to contain a few results that the user did not browse. In this way the user could deny having actively viewed any results issuing from his machine.

Clustering To make a sketch, we attempt to cluster related web pages in the user's browsing history. The keywords from the best clusters will constitute our sketch. Clustering has three parts: generating click graphs, merging click graphs, and extracting keywords.

Generating Click Graphs We can divide web viewing into sessions. Each viewing session is initiated by the user opening the web-browser or actively entering data into the web browser (e.g. entering a URL or executing a web search query). Within each session the user navigates by clicking links. For each session we can define a *click graph* $G = (V, E)$, where V is the set of web pages visited during the session and there is a directed edge between web pages p_1 and p_2 if the user clicked a link leading to the web page p_2 from the web page p_1 (see figure 2). We assume each page is labeled by its URL. In addition, there are other summary statistics associated with a page: the amount of time the user spent on the page, the number of times the user visited the page, etc..

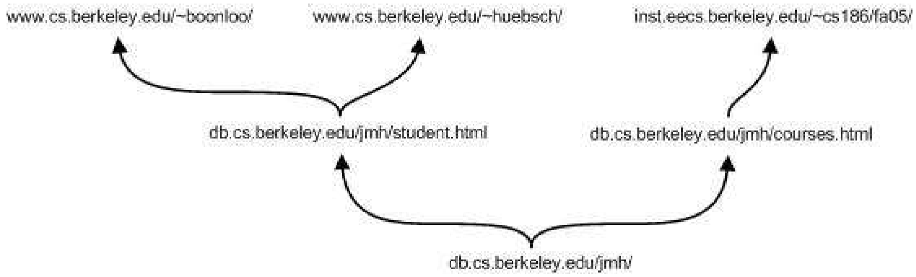

Fig. 2. A Click Graph

Merging Click Graphs We combine click graphs that pertain to sessions on related subjects. For example, when users are researching a subject, they often perform multiple queries on that subject. All these queries induce distinct click graphs, however all these graphs are related and should be combined and treated as one.

Click graphs generated from chronologically contiguous sessions are more likely to be on the same topic. Additionally, click graphs that share URLs or have many common words are likely to be on the same topic. CHORA uses these three criteria: time, word commonalities, and link commonalities, to help identify and merge click graphs that are on the same topic.

Keyword Extraction We now show how CHORA computes a vector $\boldsymbol{v}$ which contains word-score pairs $(w, s(w))$ for each word w that appears on a page in the session using a variant of TF-IDF. We can associate a duration t_p with each vertex $p \in V$ which indicates the time the user spent on that page. Let $f_p(w) = \{\texttt{\# of times w appears on p}\}/\{\texttt{\# of words on p}\}$ be the frequency of word w on page $p \in V$.[1] We define the score of word w for session i as $\hat{s}^{(i)}(w) = \sum_p f_p(w)t_p/T$ where $T = \sum_{p\in V} t_p$ is the total time spent in this cluster. It can easily be verified that $\sum_w \hat{s}_{(i)}(w) = 1$.

Finally, in order to get the words that best describe a cluster we normalize across all clusters. Let $\mathcal{I}$ be the set of clusters. $s^{(i)}(w) = \hat{s}^{(i)}(w) - \frac{1}{|\mathcal{I}|-1}\sum_{j\in\mathcal{I}, j\neq i} \hat{s}^{(i)}(w)$.

[1] In computing f_p we also take other factors into account, e.g. capitalization.

Note that not all of the $s^{(i)}(w)$s will be positive. We then score each cluster based on a combination of the time the user spent viewing pages in the cluster and the scores of the words in the cluster. The top k clusters are included in the sketch. The number of keywords included with each cluster is proportional to the score of the cluster. k is a parameter that depends on the desired sketch size.

Size of Sketch Currently, CHORA limits its DHT entries to at most 1 kilobyte. When each keyword is stored in the DHT, CHORA includes the keyword itself, the meta-data of the user's machine and as many other keywords from the same cluster as will fit. Priority is given to the most important keywords and this number tends to be around 30 in practice.

By using the estimations from [14] that each participating node in the DHT can store 1 GB, and estimating that 10% of the the users of CHORA participate in the DHT, each user can store $100,000$ keywords. In our opinion that is larger than is needed for CHORA.

Keyword Shortfalls One issue with keywords is that there are situations where it is not desirable to search over individual keywords, for example when searching for a quote. The query "It was the best of times, it was the worst of times" would almost certainly fail if we use the keyword mechanism described above.

This problem can be mitigated by asking the user to separate their query into a topic ("Charles Dickens") and a subquery (the quote mentioned above). Then CHORA would use "Dickens" as a keyword (to identify a target list of machines), but locally search each machine with the subquery. Some users might not be able to connect a quote like this with the context ("Dickens"), but we note that centralized search engines already perform exceedingly well on this type of search.

Another issue is that the DHT can only support exact matches amongst keywords. If one machine is an expert on "UC Berkeley", but a user searches for "University of California, Berkeley" the first machine may not be included in the target list.

Two ways to alleviate this problem include searching over URLs and automatically generating keywords synonyms. Sometimes a URL describes a topic better than any word. In the aforementioned example, there are many ways to describe UC Berkeley as exact keywords, but it has only one homepage: `www.berkeley.edu`. Furthermore, additional keywords can be generated on the fly, for example, by looking at the words the appear in the snippets – the 2 or 3 line query-sensitive summaries of each result – of a centralized web search.

3.2 Ranking and Aggregation

In each of the two phases, sketch query and peer query, the results must be ranked. The order of the sketch query rank is used to decide which machines to query if too many positive results come back. It is also used in the super-node model to decide where peers should be in the query tree.

Sketch query ranking Chora ranks the machines based on three criteria:

1. The number of query words that the keyword list contains.
2. Where the query words appear in the keyword list (keywords at the front of the keyword list are assumed to be more important).
3. How closely related the keywords are in the peer. (For example, if all the keywords are found in the same DHT entry, they must all be keywords for the same cluster or topic).

Peer query ranking Locally, each computer uses a desktop search application to locate the relevant web sites from their browser history. Unfortunately, desktop search systems still have poor relevancy ranking algorithms. We attempt to improve the ranking by considering the time the user spent on the page (if they spent very little time, the page is likely bad) and the number of times the user visits the page. When aggregating these results, Chora takes two things into account: the number of peers that recommend a particular site and the rank that each recommending peer gives the site (relative to their other recommendations).

The results are updated on the fly, so when Chora receives more complete responses that reorder or add additional results, these updates are immediately conveyed to the user. Ideally these results would be used to rerank typical web search engine results. In our current implementation we place the Chora results beside the web search engine results.

4 Evaluation

Keyword Coverage Consider k users, each with a set D_i documents. Suppose we have a query q. Clearly if we issued q against a full-text index which indexed every word for each document $d \in D_1 \cup D_2 \cup \ldots \cup D_k$ then we would be absolutely sure that the results $R_f(q)$ (f for full) cover every possible document. In our setting, the DHT acts as a *partial-text-index*. If $R_p(q)$ (p for partial) is the results for querying q against our partial text index, then we are interested in maximizing the coverage ratio, $C(q) = |R_p(q)|/|R_f(q)|$. Coverage ratio is a relaxed definition of recall: it says that the set of relevant documents *is* $R_f(q)$. In this relaxed setting, precision is trivially 1.

One way of growing $R_p(q)$ is to just store more keywords for each document. In the limit, we approach a full-text index. However, we tradeoff increasing keywords with the increased storage and communication overhead.

One observation is that as long as we use a partial-text index, one will always be able to construct a query q that makes $C(q)$ small [2]. However, in reality, it is only important that $C(q)$ be large for *reasonable* queries. For a given topic, like "Peer to Peer Networks," we would expect a reasonable query to be something like "Gnutella Ultrapeers," whereas there are a huge number of unreasonable

[2] Note that this analysis does not apply if we assume that users will change their querying behavior to first specify a topic (e.g. "Charles Dickens") that is used to get a list of machines, and then specify a query.

queries. Our ultimate goal is to cover a high percentage of the reasonable queries for all topics with a small number of keywords for each cluster in the sketch.

To test our system, we hand-generated 10 topics. You can see one such topic in Table 1. The table contains a set of training and test queries for the popular topic, “Harry Potter.” Each topic has 15 training queries that describe the topic, and at least 5 test queries.

When users index the Google results for training queries like “Harry Potter” and “Professor Snape,” we expect that one or more of their partial text indices would be able to answer test queries like “Ron Weasley.”

Given this, we now create a set of fictional users. Each user u chooses a topic from their topic distribution T_u, and then from within that topic they choose k queries from the training set. They then take m of the results for each query from Google as their own pages. At this point, each user has km webpages.

We then build a full-text-index over all the documents, as well as the Chora keyword-list sketches. For each topic set, we run the test queries against each index and compute the coverage for each query.

The coverage percentages in Table 1 indicate that for the training queries, the coverage is perfect, which is not surprising. The test query coverages are indicative of a trend across all the topics: we found that queries either had full coverage ($R_p = R_f$), or no coverage at all ($R_p = 0$). The keyword generation mechanism, while good in some instances, nevertheless fails for many queries we consider to be reasonable extensions of the training sets [3].

Harry Potter	
Training Queries	**Percent Coverage**
Harry Potter	100 %
Harry Potter and the Sorcerer's Stone	100 %
Professor Snape	100 %
Professor McGonagall	100 %
Albus Dumbledore	100 %
...	
Test Queries	**Percent Coverage**
Hogwarts School of Magic	100 %
Phoenix hair wand	100 %
Ron Weasley	0 %
Daniel Radcliffe	0 %

Table 1. Training and Test Queries for the topic Harry Potter, along with their coverage

To understand the coverage across topics, Figure 3 was generated by taking the average coverage over each topic and then placing each average in the indicated bins. While it's encouraging that on all topics most test queries are hit, there clearly remains room for improvement.

[3] Again note that if the query here were topic = “Harry Potter”, subquery = “Ron Weasley” we would do much better

Because we ran this experiment on hand-picked data and did not actually browse the web pages, it does not make sense to talk about a relevant set of documents that is a subset of $R_f(q)$. We first hope to improve the coverage ratio and then run our algorithms on real user data where users can describe a relevant set of documents for the query, and hence we can generate true precision and recall numbers.

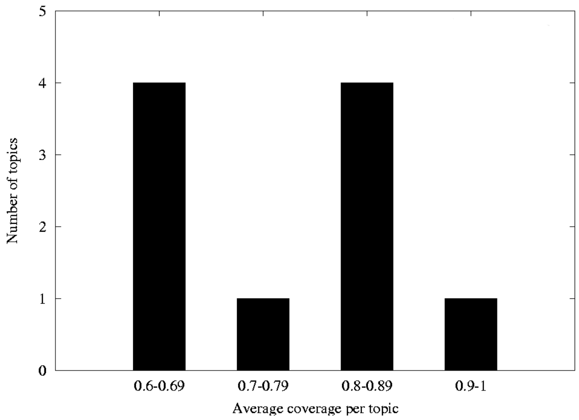

Fig. 3. Average Topic Coverage for Test Queries (each user has at most 1000 keywords per topic)

5 Related Work

While several P2P Web search engines have been created to search the entire web, [1–3] and to a lesser extent [4], Chora is more closely related to [2, 15].

Similar to Chora, [2] proposes a two-phase paradigm. However, these two phases are significantly different to Chora's. The first phase looks for peers by searching for both query terms and the user's bookmarks in a full inverted index. It then uses the KL divergence over the distribution of words to prune the list of candidates. At query time, the query is sent point-to-point to all peers discovered during step 1. In contrast, Chora exploits the user's full web viewing history and habits to generate (and prune) the keywords which summarize the data on the computer, and distributes the query through an optimized query tree. In addition, Chora uses the peers' web viewing history to recommend documents.

Other approaches have been proposed for finding peers with related content in P2P systems. In [16], the authors of the Minerva system analyze multiple algorithms that can be used to compare a query to a peer's local database. Unlike Chora, the Minerva analysis does not divide the database of documents into clusters or click graphs, but instead looks at global statistics of terms when choosing the best peer to query. However, like Chora, Minerva generates lists of the best peers to query by storing term-to-peer mappings in a DHT [17]. In [18], the system routes queries to peers using an implicit, distributed index, rather than an explicit index as in CHORA. For each neighbor, a node stores a summary of the content available at that neighbor and all nodes within a small number of hops of that neighbor. A query is routed to the neighbor with the most related summary.

Coopeer is similar to CHORA in the sense that it does not necessarily attempt to replicate and improve the centralized web search, but, initially, at least, to complement it. It can be seen as a web-search generalization of Gnutella. While it uses local flooding for queries, it attempts to learn a good network on which to do this flooding. It is also similar to CHORA in two significant ways. 1) Limited information, besides query results, are removed from a users' computer. 2) It attempts to use information from the users' web favorites to answer the query. However, CHORA differs from Coopeer significantly in that CHORA uses a two-step query while Coopeer floods the network. Also, while Coopeer only use favorites, CHORA makes full use of the web browsing history, and attempts to organize it according to topics.

As previously mentioned, CHORA uses some techniques developed in the Gnutella setting, such as the concept of ultrapeers [19, 20] when building the query tree (see Section 2 for details). Another similarity is that CHORA's two-phase query processing can be seen as a dynamic version of semantic overlays [21], which limit the flood plain of queries in Gnutella. Each semantic overlay clusters users around a particular topic and when a file is looked for, only the relevant semantic overlays are flooded.

6 Conclusions

We have presented CHORA, a P2P Web Search engine that complements, not replaces, traditional web search engines. The execution of a query in CHORA involves a two-step paradigm, where first a set of peers is selected to query based on summary sketches, and then the query is executed on these peers in an ad-hoc query dissemination tree. The sketches are created using a sketch generation algorithm which employs the novel use of a click graph.

CHORA could be further improved by integrating techniques from the IR community into the sketch generation algorithm. Also, it could be augmented by using learning techniques. For example, CHORA could be customized to each peer by weighting the advice of peers that it has, in the past, found helpful. Another example is that keywords which are commonly searched for could be pinned into the list of keywords for a user and unused keywords could be replaced by other relevant words. We hope to explore these areas in future work.

Acknowledgements We would like to thank Joseph Hellerstein and Timothy Roscoe for their helpful conversations and support.

References

1. Festa, P.: Search project prepares to challenge google. C-net News (2001)

2. Bender, M., Michel, S., Weikum, G., Zimmer, C.: Bookmark-driven query routing in peer-to-peerweb search. In: Workshop on Peer-to-Peer Information Retrieval. (2004)
3. Christen, M.: (Yacy - distributed p2p-based web indexing) http://www.yacy.net/yacy/.
4. Suel, T., Mathur, C., Wu, J.W., Zhang, J., Delis, A., Kharrazi, M., Long, X., Shanmugasundaram, K.: Odissea: A peer-to-peer architecture for scalable web search and information retrieval (2003)
5. Stoica, I., Morris, R., Karger, D., Kaashoek, F., Balakrishnan, H.: Chord: A scalable Peer-To-Peer lookup service for internet applications. In: Proceedings of the 2001 ACM SIGCOMM Conference. (2001) 149–160
6. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content addressable network. Technical Report TR-00-010, Berkeley, CA (2000)
7. : CLIP2: Gnutella Protocol Specification v 0.4 (2001)
8. Loo, B.T., Huebsch, R., Stoica, I., Hellerstein, J.M.: The case for a hybrid p2p search infrastructure. Technical report, Intel Research (2003)
9. : (Gnutella proposals for dynamic querying.) http://www9.limewire.com/developer/dynamic_query.html.
10. : (Google desktop) http://desktop.google.com.
11. Rhea, S., Godfrey, B., Karp, B., Kubiatowicz, J., Ratnasamy, S., Shenker, S., Stoica, I., Yu, H.: Opendht: A public dht service and its uses. (In: Proceedings of ACM SIGCOMM 2005)
12. Huebsch, R., Hellerstein, J.M., Boon, N.L., Loo, T., Shenker, S., Stoica, I.: Querying the internet with pier. In: Proceedings of 29th International Conference on Very Large Databases (VLDB). (2003)
13. Dabek, F., Cox, R., Kaashoek, F., Morris, R.: Vivaldi: A decentralized network coordinate system (2004)
14. Li, J., Loo, B.T., Hellerstein, J.M., Kaashoek, M.F., Karger, D.R., Morris, R.: On the feasibility of peer-to-peer web indexing and search. In: 2nd International Workshop on Peer-to-Peer Systems. (2003)
15. Zhou, J., Li, K., Tang, L.: Toward a fully distributed p2p web search engine. (In: Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems)
16. Chernov, S., Serdyukov, P., Bender, M., Michel, S., Weikum, G., Zimmer, C.: Database selection and result merging in p2p web search. In: Third International Workshop on Databases, Information Systems and Peer-to-Peer Computing. (2005)
17. Bender, M., Michel, S., Triantafillou, P., Weikum, G., Zimmer, C.: Minerva: Collaborative p2p search (demo). In: Proceedings of the 31st International Conference on Very Large Databases (VLDB). (2005)
18. Petrakis, Y., Koloniari, G., Pitoura, E.: On using histograms as routing indexes in peer-to-peer systems. In: Second International Workshop on Databases, Information Systems and Peer-to-Peer Computing. (2004)
19. : (Gnutella ultrapeers) http://rfc-gnutella.sourceforge.net/Proposals/Ultrapeer/Ultrapeers.htm.
20. : (Gnutella) http://gnutella.wego.com.
21. Crespo, A., Carcia-Molina, H.: Semantic overlay networks for p2p systems. Technical report (2002)