Routing queries through a peer-to-peer InfoBeacons network using information retrieval techniques

Sangeetha Seshadri and Brian F. Cooper
College of Computing, Georgia Institute of Technology

***Abstract*— In the InfoBeacons system, a peer-to-peer network of *beacons* cooperates to route queries to the best information sources. Many internet sources are unwilling to provide more cooperation than simple searching to aid in the query routing. We adapt techniques from information retrieval to deal with this lack of cooperation. In particular, beacons determine how to route queries based on information cached from sources' responses to queries. In this paper, we examine alternative architectures for routing queries between beacons and to data sources. We also examine how to improve the routing by probing sources in an informed way to learn about their content. Results of experiments using a beacon network to search 2,500 information sources demonstrates the effectiveness of our system; for example, our techniques require contacting up to 71 percent fewer sources than existing peer-to-peer random walk techniques.**

Categories and Subject Descriptors: H.3.4 [Information Storage and Retrieval] Systems and Software – distributed systems, information networks, performance evaluation; H.3.3 [Information Storage and Retrieval] Information Search and Retrieval – search process; C.2.4 [Computer-Communication Networks] Distributed systems – Distributed databases, distributed applications
Additional Keywords: peer-to-peer systems, information search and discovery

I. Introduction

There is an explosion of useful data available from dynamic information sources, such as "deep web" data sources, web services, web logs and personal web servers [5]. The Internet and web standards make it possible and easy to contact a source and retrieve information. But the proliferation of sources creates a challenge: how to find the right source of information for a given query? Peer-to-peer search mechanisms are useful for finding information in large scale distributed systems, but such mechanisms often rely on the explicit cooperation of information sources to export data, data summaries or data schemas to aid in searching. Many data sources are unwilling to provide this cooperation, either because they do not want to export valuable information, or because they do not want to modify their service software and expend the resources necessary to cooperate with a peer-to-peer system.

How can we build a peer-to-peer system that is useful for searching large numbers of distributed sources when those sources will not provide more cooperation than simple searching? Our approach is to build a network of peers, called *InfoBeacons*, that are loosely-coupled to the information sources: beacons connect to sources and utilize their existing search interface, but do not expect tight schema integration, data summary export, or any other high-level cooperation from the source. InfoBeacons act to guide user keyword queries to information sources, where the actual processing of queries is done, and then retrieve results and return them to the user. As such, the InfoBeacons network is similar to a super-peer network [37], except that the beacons do not expect the same level of cooperation from sources (i.e. exporting their indexes or schemas) that super-peers expect. In order to choose appropriate beacons and sources for a given query, we adapt techniques from networked information retrieval [17], [15]. These techniques allow us to predict how good a source or beacon will be for a given query. As a result, we can route queries through the system to the most appropriate sources, and avoid overburdening beacons and sources with irrelevant queries.

In this paper, we describe how the InfoBeacons system uses IR techniques to perform query routing. Beacons maintain a cache of the information available at other beacons and sources and use this cache to determine where to route queries. Each beacon is responsible for a subset of sources (to reduce the load on any individual beacon), and many beacons cooperate to route queries among a very large number of sources. We examine three approaches to inter-beacon routing. In the *hierarchical* approach, a "superbeacon" uses IR-style ranking to choose among beacons. In the *flat* approach, beacons treat each other as regular sources, forming a flat network composed of both beacons and sources. A beacon routes queries to the most promising "neighbor," which may be a source or another beacon. The third approach is a *hybrid* approach that combines the hierarchical and flat architectures, by arranging beacons in two levels with multiple superbeacons and multiple "leaf" beacons clustered under each superbeacon. Superbeacons route queries to leaf beacons and other superbeacons, while leaf beacons route queries to sources.

When a source returns results for a query, the beacon caches those results to aid in future routing decisions. However, the decentralization in the system means that each beacon sees a limited number of queries, and thus the cache may not warm very quickly. Moreover, a simple beacon cache usually has a limited view of the full range of content available at a source. In order to accelerate the warming of the beacon's cache and expand its coverage of a source's data, we use *query probing*: proactively querying a source with keywords to sample the source's data [10], [19]. Generating good probe queries is central to the performance of query probing. We present a technique called *informed probing* that utilizes the information available on a source's crawlable web interface to generate effective probe queries. Improving the quality of the cache results in improved query routing.

We have experimentally evaluated our techniques with real Internet data using a prototype implementation of the InfoBeacons system. Experiments demonstrate that our techniques perform better than random walks, an efficient and scalable peer-to-peer routing mechanism [26], [1]. We also evaluated our query probing technique using the prototype implementation of our system. With

Brian Cooper is now at Yahoo! Research. Current email: cooperb@yahoo-inc.com. Sangeetha Seshadri's email is sangeeta@cc.gatech.edu.

as little as a single query, informed probing is able to extract as many documents as a large number of random dictionary-word-based queries [10].

There are many existing techniques for performing information discovery in peer-to-peer systems. However, these techniques must be adapted, and new techniques must be developed, to deal with dynamic, uncooperative information sources. First, many systems rely on the explicit cooperation of the data sources to export content or content summaries [37], [30], [29], [33]. Unfortunately, deep web sources are often unwilling to do so, and our system must use special approaches to learn about a source's data. Second, some systems gather information about sources in order to build a static, federated search system [24]. Many data sources are dynamic, frequently changing their content or appearing and disappearing, and we must dynamically adapt our routing based on the most current information. Third, many systems focus on locating documents based on their identifier [31] or on keywords in document metadata (such as the title) [22]. Our work focuses on full text content-based searching and routing, which presents new performance challenges. More related work is discussed in Section V.

In this paper, we examine how information retrieval techniques can be adapted to route queries through the InfoBeacons system. Specifically, we make the following contributions:

- We present and compare the *hierarchical*, *flat* and *hybrid* architectures for routing queries between beacons.
- We describe an effective query probing technique, *informed probing*, that uses a minimal number of *informed probes* to extract content from data sources and accelerate warming-up of beacon caches.
- We present an experimental evaluation, which demonstrates the performance tradeoff between different routing architectures. The results also show that our techniques can significantly outperform previously reported approaches.

This paper is organized as follows. First, in Section II we describe how beacons cache source data, and use those caches to route queries. Then, in Section III we describe and compare qualitatively the hierarchical, flat and hybrid architectures. Next, in Section IV we present experimental results quantitatively evaluating our techniques. We examine related work in Section V, and discuss our conclusions in Section VI.

II. Routing queries using cached results

In the InfoBeacons system, *beacons* connected in a peer-to-peer network work together to guide user queries to useful information sources. Beacons accept user keyword queries, connect to sources, submit the queries, retrieve results, and return them to the user. The user is therefore shielded from the complexity of choosing and searching many different sources. Many beacons, each managing multiple information sources, are needed to scale to a large number of information sources.

Figure 1 shows the architecture of an individual beacon. The *user query interface* provides an API for users to submit queries and retrieve results. Currently, our prototype accepts queries as HTTP GET requests and returns results encoded in XML. User queries in our system are sets of multiple terms, although our techniques can be extended to deal with other query types (such as general boolean queries). The *cache* contains partial information about the content available at sources and other beacons, and

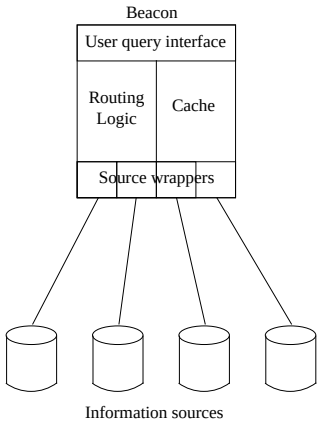

Fig. 1. Beacon architecture.

the *routing logic* uses the information in the cache to determine where to route queries. Lightweight *source wrappers* submit queries to data sources and retrieve results. Our current prototype includes simple wrappers for submitting queries via HTML forms and screen-scraping the results. Techniques for creating more complex wrappers, and creating wrappers automatically, have been examined by others [34], [11], [2] and can be integrated into our framework.

Beacons can be run by libraries, universities, ISPs, corporations or any organization that wants to provide searching services to its user group. In order to reduce the resource requirements for these hosts (and to encourage them to participate), beacons are designed to be lightweight components. In particular, our goal is to minimize the memory, processing and bandwidth requirements for running a beacon. We address this goal partially through implementation techniques (for example, by designing the in-memory data structures to be very compact) and partially through algorithms for efficiently routing queries (so that only "relevant" beacons should have to process a given query, reducing the load on other beacons). In this paper we focus on developing such efficient routing algorithms. Because we expect beacons to be run by libraries, ISPs and so on, and not necessarily by end users, beacons will typically be long-lived peers, even if the information sources they manage are not. Thus, while the system is robust to beacons joining and leaving, we expect such turnover among the beacons to be infrequent.

Each beacon is responsible for a small number of sources (say, 100 or so). Since it is too expensive to send every query to every source, the beacon must determine the most appropriate sources for each query. Ideally, each source would export a summary of its content to help the beacon route queries. However, many Internet information sources are willing to accept queries and return results, but are unwilling to provide more cooperation by exporting their contents, content summaries, or schema information. This is what we mean by "uncooperative sources." As a result, a beacon must learn which sources are good for each query, while relying only on the sources' basic search interface. Beacons learn about sources by caching results from previous queries, and then use these results to choose appropriate sources for future queries. We say that the beacon is *loosely coupled* to the information sources. This loose coupling ensures that it is cheap to integrate a new source, so that the system is tolerant to frequently appearing and disappearing sources.

A beacon is like a *networked information retrieval system* such as GlOSS [17] or CORI [15], but adapted to work in a peer-to-peer manner with uncooperative sources. First, the beacon is loosely coupled to the information source, allowing it to work with even uncooperative sources, while many existing networked IR systems require sources to export their contents or content summaries. The high turnover of data sources observed in many peer-to-peer systems also makes loose coupling important, as mentioned above. Second, existing networked IR systems are usually centralized, while in our system, sources are managed in a decentralized way by many beacons, enhancing the scalability of the system. Third, existing systems usually build up a static characterization of the source contents, while a beacon constantly adapts its cache in response to new results. Experiments show that continual adaptation improves source selection performance (especially when content is changing frequently) [13].

A. ProbResults routing

We have developed a function, called *ProbResults*, to determine where to route queries based on the information in the beacon's cache. ProbResults uses the cache to predict the number of results that a source will return containing the given query words; this predicted number is called the *ProbResults score*. The ProbResults function uses several values:

- n_Q: the number of terms in the query Q
- R_i^s: the number of past results from source s that contained query word i
- tq_s: the total number of times that source s has been queried by the beacon

The ProbResults score for site s for a query Q is calculated by:

$$\textit{ProbResultsScore}_Q^s = \prod_{i=1}^{n_Q} \frac{R_i^s}{tq_s} \quad (1)$$

Each R_i^s/tq_s term represents the expected number of results (for any query) from s that contain query word i. Multiplying the R_i^s/tq_s values produces an aggregate score for all of the query words.

In order to keep the beacon lightweight, the beacon cache does not contain whole documents, but instead only retains statistics about the word distributions in the results returned from each source. In fact, the only information that is needed for each source s is the R_i^s value for each word and the tq_s value for the source. The result is that the beacon cache is very compact, and in our experiments a beacon responsible for 100 sources needed only a few tens of megabytes of cache. This caching structure is adapted from the source summary structure used in the GlOSS system [17]. Moreover, since the beacon only updates this term count data structure for retrieved documents, and does not have to build a full inverted index, query results can be processed very quickly (in fact, more quickly than they can be downloaded from the data source[1]).

Consider two sources s_1 and s_2 that are managed by the same beacon B. Source s_1 contains chemistry papers, while s_2 contains retail customer survey responses. After several queries, a portion of the beacon cache might contain:

[1]For example, the time to download 3,394 web pages from www.uga.edu to Georgia Tech was 173 seconds, while the time to parse and cache those documents on a 2.8 GHz Xeon machine was 12 seconds.

	exothermic	oxygen	reactions	product	consumer	tq_s
s_1	70	80	120	0	40	100
s_2	10	15	80	130	210	150

The numbers in this table represent the R_i^s counts for each word and source. Now imagine that a user submits a query for "exothermic reactions" to B. The ProbResults score for s_1 is $(70/100) \times (120/100) = 0.84$, while the score for s_2 is $(10/150) \times (80/150) = 0.036$. Thus, the beacon B would first contact s_1 to search for "exothermic reactions." This makes sense, since site s_1 contains chemistry literature, and the beacon cache reflects that more previous results from s_1 contain "exothermic" and "reactions" than those from s_2. On the other hand, if the user searches for "consumer reactions," we would expect s_2 to receive a higher ProbResults score, and it does, scoring 0.75 (compared to 0.48 for s_1).

The ProbResults function is adapted from the Ind metric used in the bGlOSS information retrieval system [17]. ProbResults differs from Ind in several key ways in order to work in a loosely-coupled, dynamic peer-to-peer architecture. First, ProbResults tries to characterize both the behavior and the content of a source, while Ind focuses only on the content. For example, the R_i^s value used by ProbResults counts documents once per time they are returned as a query result, not just once overall (as in Ind). Thus, ProbResults gives higher weight to documents that are returned multiple times, better characterizing the behavior of the source in response to queries. Characterizing a source's behavior helps compensate for the inexact picture a loosely-coupled beacon has of the source's content. Another difference is that both the tq_s and R_i^s values are constantly updated in the beacon cache, unlike in GlOSS, where a static source summary is constructed. As a result, ProbResults produces scores that are tuned to the current behavior of the source, unlike Ind, whose scores can become stale over time.

Although a source may return documents that contain a particular word, actual queries containing the word may produce no results at that source. This may either be due to frequent content change at the data source or the fact that the word is not considered a useful query term by the source. For example, although most documents at a weather related source may contain the term "weather", the source may produce no results when simply queried for "weather", as the term is very common and hence carries no weight. We have developed a technique called *experience weighting* that allows the cache to dynamically adapt and align itself with the actual behavior of the information source. Experience weighting uses an "experience factor" $EF \geq 1$. After each query, the beacon multiplies the cache count for query terms for that source by *EF*, if the source returns results for the query; otherwise, the cache count is divided by *EF*. Thus, over time, experience weighting adjusts the term counts in the cache to reflect not just the summary of data available at the source, but also the actual *behavior* of the source. Moreover, experience weighting allows the beacon to deal with dynamic information at sources. If a source changes its content, invalidating the cached information and causing the beacon to make bad routing decisions, the cached values will quickly be experience weighted downward to adjust the routing. In practice, experience weighting is more effective at adapting the beacon cache to reflect the current contents and behavior of sources than other alternatives (such as a *forgetting factor*) [13].

The effectiveness of the ProbResults function depends on the quality of the cached information. If the cache is biased toward

a particular topic or popular documents, queries may not be routed to sources with relevant content. To deal with this problem, we employ several techniques. First, we use a technique called *informed probing* (described in the next section), to add a variety of content to the beacon cache, not just the most popular content. Second, if the cache contains no information about a given query term, we do not assume that the source has no content relevant to the query term. Instead, we use a special constant, P_{min}, instead of R_i^s/tq_s, when R_i^s (or tq_s) is zero. In practice, a small value of P_{min}, such as 0.0001, works well. The P_{min} constant also allows us to calculate a non-zero ProbResults score for a source if we have cached information from the source about some, but not all, of the query terms. Since Equation (1) is a product, without P_{min} the ProbResults score would be zero if there were any query terms that matched no cached documents for a given source. Third, when ejecting entries from the cache, we preferentially eject entries that have little information value; that is, entries whose cache count are close to P_{min}. This retains high quality cached information even if queries tend to bias the cache. Fourth, even if a query is for rare or less popular content, the beacon network will continue searching until results have been found. Such queries will be more costly than queries for popular content, but they will still find information if it exists in sources managed by beacons. Finally, note that a highly skewed workload (i.e. where queries follow a Zipfian distribution) may bias the cache toward the frequent queries, but that this in fact results in good performance for frequent queries, causing an overall improvement in beacon performance compared to a non-skewed query workload.

We could also add randomization to the ProbResults function, to improve the chance that a source which appears irrelevant due to cache bias is still contacted by the beacon. Of course, more randomization means that more truly irrelevant sources are contacted by the beacons, reducing the efficiency of the system. Similarly, we could add personalization weights to ProbResults, to assist in answering queries that were for a specific topic area not necessarily well represented by the beacon cache. We have not implemented such randomization or personalization, but we note that the beacon framework is extensible enough to support such techniques.

Experimental results in [13] show that our optimized ProbResults function produces better predictions for our application scenario than the Ind ranking or other ranking functions used in the GlOSS and CORI systems. For example, in a system where uncooperative source contents are frequently changing, beacons using ProbResults contact about half the sources compared to beacons using the other ranking functions in order to find the same quality of results. There may be other ranking functions that result in better predictions for different scenarios. However, we have found in practice that ProbResults works very well for our loosely-coupled network of highly dynamic data sources.

B. Warming beacon caches with informed query probing

In the InfoBeacons system, query routing is based on source summaries constructed incrementally by caching results of earlier queries. However, relying only on user queries to construct summaries makes cache warm-up a slow process, especially since the decentralization in the system means that each beacon sees a limited number of queries. Moreover, purely reactive caching may provide incomplete coverage of a source's data, since the cache may be biased towards the results of popular queries and may not have any information matching less popular queries. The beacons can proactively construct initial data source summaries by sending a small number of probe queries to the data sources. This probing can accelerate the cache warm-up process, and help to ensure wider coverage of a source's content. Since beacons handle uncooperative data sources, some of which may impose limitations on the number of requests serviced, it is important to use as few probe queries as possible. Hence, the challenge here is to retrieve *maximum* information from a source while using a *minimum* number of probe queries.

Query probing has been proposed by other investigators as a way to deal with uncooperative sources [10], [19]. Typically, query probes may be terms randomly chosen from a dictionary, or terms from a rule-based classifier for determining the topic of a source. However, in each case a considerable number of queries may have to be issued to the source before any results are retrieved. For example, during our experiments, we observed that on average only one percent of random query probes returned any results from sources. The problem is that the query prober has no *a priori* information about the source, and must try many probes before finding one that is relevant to the source's content.

Our approach is to use probes that are tailored to each source, to improve the chance that each probe retrieves information. Our probing strategy, called *informed probing*, exploits the fact that the information available on a data source's crawlable web-interface (e.g., HTML form) is representative of the content at the data source, even if the content itself is not crawlable. For example, a deep web data source containing medicine-related information is likely to have keywords relevant to medicine in the metadata tags (⟨TITLE⟩, ⟨META⟩, etc.) and page text of the crawlable web interface. We therefore construct probe queries from these keywords. The probability that probes using these terms will return results is much higher than that of probes based on randomly chosen dictionary words.

To construct probes, we take terms preferentially from ⟨TITLE⟩ and ⟨META⟩ tags to construct a query with n terms. For example, we might construct probes with $n = 10$ terms. If there are less than n terms in the ⟨TITLE⟩ and ⟨META⟩ tags, we must select some terms from the text of the page. We can use a weighting mechanism (such as TF/IDF, where IDF is computed over a corpus of web pages) to rank the terms in the text and choose the highest weighted terms for inclusion in the probe.

Experiments in Section IV evaluate both the standalone performance of the query probing technique and its impact on overall system performance. Our results show that informed probes are very effective at extracting source data. Our approach could be extended by augmenting informed probes with synonyms to further increase the probability of finding matching content, although this extension is not implemented in our current prototype.

III. Routing queries between beacons

Different sources contain widely varying information, and a single beacon may not have the right sources to answer a given query. Even though a user initially submits his query to a single beacon, that beacon may have to forward the query to several other beacons in order to retrieve results. The simplest approach would be for the beacon to send the query to all of its neighbor beacons, but this flooding approach is too expensive in a large

scale system. In this section we examine how a beacon can intelligently route queries to other beacons.

One approach is to use existing peer-to-peer routing techniques. For example, a beacon could forward each query to a randomly selected neighbor. Such "random walks" [26], [1] have been shown to be an effective and scalable way of routing queries in a peer-to-peer network where there are many possible results for a given query. However, no content information is used during the routing process, and such information could be used in routing to reduce the number of contacted peers while still returning high quality results. Other existing techniques focus on locating documents based on their identifier [31] or on keywords in document metadata (such as the title) [22]. Our goal is to build a system that can effectively search the full text of documents.

Our approach is that beacons use the ProbResults ranking to route queries to sources and other beacons. In particular, we study three mechanisms for routing queries between beacons:

- *Hierarchical*: A "superbeacon" caches results from beacons, and uses this cache along with ProbResults to choose beacons for a given query.
- *Flat*: Each beacon's neighbor beacons are treated as regular sources, and ProbResults produces a single ranking of both information sources and neighbor beacons.
- *Hybrid*: Beacons are organized into a hybrid, two-level network with superbeacons and leaf beacons. The superbeacons cache results from leaf beacons, which themselves cache results from data sources. The beacons in the superbeacon level are organized as a flat network.

A. Hierarchical network

An example of the *hierarchical* approach is shown in Figure 2. As the figure shows, the superbeacon is connected to the rest of the system's beacons, who are in turn connected to the system's sources. Each query is submitted to the superbeacon, which uses ProbResults to rank the beacons for that query. The superbeacon routes the query to beacons in decreasing order of ProbResults score, until it has received enough results to satisfy the user's threshold. As with regular beacons, the superbeacon caches the results it receives for use in routing future queries.

One alternative approach we examined was to have each beacon send a copy of its cache to the superbeacon, and have the superbeacon evaluate the query against each cache to determine which beacon is best. This approach may result in more accurate routing, since the superbeacon would have more information about each beacon. However, while each beacon's cache is small, in a large scale system there are likely to be many beacons, and a large amount of space would be required to store a copy of every beacon's cache. Our goal is to keep beacons, even the superbeacon, as light-weight as possible, and therefore it is infeasible to expect the superbeacon to store copies of all of the beacon caches. As a result, we chose the approach described above, where the superbeacon keeps its own compact cache of results from the beacons and uses ProbResults to perform the routing.

B. Flat network

Unfortunately, the hierarchical approach still may not be scalable enough. The superbeacon must know about all of the beacons in the system, and must perform processing on every user query. This degree of centralization is contrary to the decentralized philosophy of peer-to-peer systems, since the superbeacon can quickly become a bottleneck hindering the performance of the system. The superbeacon can also become a single point of failure; if the superbeacon fails, the InfoBeacons network will be effectively unavailable until a new superbeacon is selected. Even when the new superbeacon is chosen, its cache will be cold and there will be a period of inefficient searching as its cache warms up.

A more scalable and robust approach is to maintain the routing information in a decentralized manner, which is the goal of the *flat* architecture, as shown in Figure 3(a). As this figure shows, a beacon's neighbors consist of both information sources and other beacons, forming a one-level "flat" network. Each beacon caches results both from information sources and from other beacons. For each query, ProbResults is used to produce a single ranking of neighbors, and these neighbors are contacted in order of decreasing ProbResults score until enough results have been found. For example, a beacon might first route the query to an information source with a score of 0.9, then to a neighbor beacon that has a score of 0.8, then to another information source with a score of 0.7, and so on.

The flat approach avoids the centralization of the hierarchical approach, since there is no beacon that has to process every query or know about every other beacon. A disadvantage of the flat approach is that each beacon has less information than the superbeacon, and thus prediction accuracy may suffer.

In experiments with our beacon prototype, we found that the structure of the flat network had a large impact on performance. Initially, we constructed a random topology, connecting each beacon with a randomly chosen set of beacon neighbors. An example of this topology is the flat network shown in Figure 3(a). In this topology, a given beacon has a path to all of the other beacons (and sources) along each of its beacon neighbor links. This means that the same documents can appear as results from any of these neighbor links, and, after a while, the ProbResults ranking begins to assign the same score to all of the beacon neighbors. This prevents the beacon from making effective routing decisions and performance suffers.

If we instead use a spanning tree structure, the inter-beacon routing performs better. An example is shown in Figure 3(b). A distinct set of beacons and sources is reachable along any given beacon neighbor link. The result is that the beacon's ProbResults scores effectively distinguish between the information available along each of the neighbor links, improving routing accuracy. Results in Section IV demonstrate the performance improvement of the spanning tree structure.

Beacons form themselves into a spanning tree in a decentralized manner. Since we are not trying to form a "minimum" spanning tree, this process is relatively straightforward. When a new beacon B joins the network, it connects to one other existing beacon, and designates the existing beacon as its "parent beacon." This designation of a "parent" is only used to ensure a spanning tree topology, and does not denote any difference in functionality between the beacons; in particular, the connection is not directed (unlike in the hierarchical architecture) and queries flow both ways across the connection. The beacon B may receive other connections as other beacons join the network, but will always have only one parent beacon. Every beacon in the network has

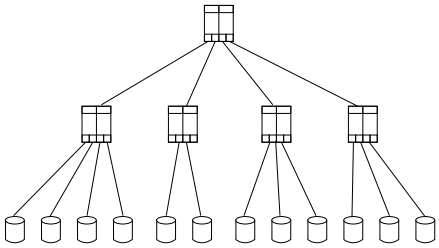

Fig. 2. Hierarchical architecture.

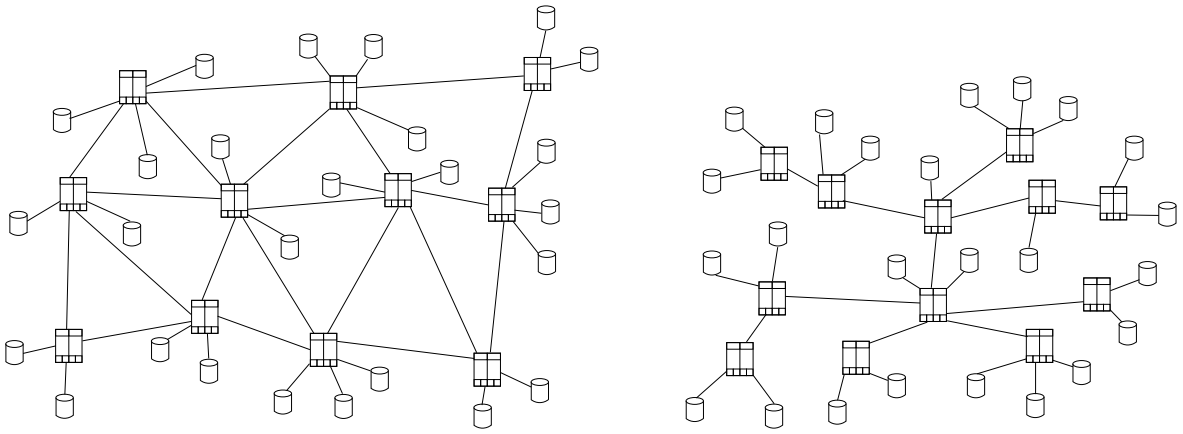

Fig. 3. Flat architectures: (a) random, and (b) spanning tree.

one parent, except the first beacon to be created, which has no parent (and which can be thought of as the "root" of the tree). The result is that the network always maintains a tree structure. If a beacon becomes disconnected from its parent, it rejoins the network, choosing another parent. We do not expect a high level of "churn" among the beacons themselves (e.g., beacons frequently joining and leaving), since a beacon is created with the intent of serving as a long-running information service. Thus, while there may be temporary disconnections, most of the time the spanning tree should remain connected despite the sparseness of the connectivity graph.

One disadvantage of the flat topology is that if there are very many beacons, there could be long delays as queries travel long overlay paths (even if the routing is effective.) For very large networks, then, the hierarchical or hybrid network (described next), which limit the number of beacons contacted, would provide lower delays.

C. Hybrid network

The hybrid architecture bridges the extremes of the hierarchical and flat architectures. Figure 4 shows an example hybrid network. In this architecture, beacons are organized into two levels, with each beacon at the *superbeacon* level responsible for a distinct set of beacons at the *leaf* level. User queries are processed by the superbeacons and data sources are managed by the leaf beacons. While routing between superbeacons and leaf beacons follows the hierarchical approach, routing between superbeacons follows the flat architecture.

The hybrid architecture embodies a tradeoff between the robustness and scalability of the flat architecture, and the searching efficiency of the hierarchical network. Like a flat architecture, there is no single superbeacon that must handle all of the queries in the system. Of course, each superbeacon in a hybrid network will be more loaded than an average beacon in a flat network. However, if superbeacons become overloaded we can promote more leaf beacons to superbeacon status to help share the load. Thus, unlike the hierarchical network, we can dynamically adjust the amount of work done by superbeacons based on the current load in the system, enhancing scalability. Similarly, like the flat architecture, the hybrid architecture has no single point of failure. The failure of any one superbeacon does not make the whole searching system unavailable, although several beacons will be unavailable until they can choose or connect to a new superbeacon. Like a hierarchical architecture, superbeacons in the hybrid network enable more efficient searching than the flat network, since superbeacons see more queries and have more information about the system than an average beacon in the flat architecture. Of course, since there are multiple superbeacons in the hybrid network, each will have less information than the single superbeacon in the hierarchical network.

In fact, both the flat and hierarchical architectures can be seen as special cases of the hybrid architecture. A hybrid network with one superbeacon is a hierarchical network, while a hybrid network where every beacon is a superbeacon (and the superbeacons connect directly to sources) is a flat network. Therefore, by changing the number of superbeacons in the hybrid network, we can make the system more or less like a hierarchical or flat network, depending on the current needs of the system.

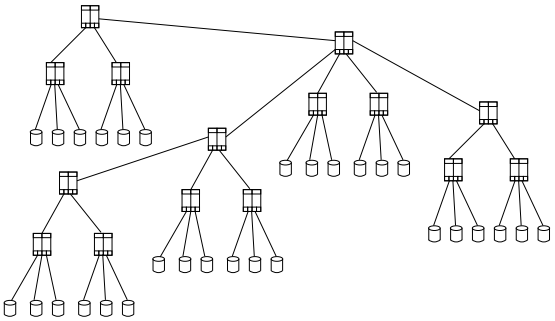

Fig. 4. Hybrid beacon network architecture.

Sources	2,500
Total data size	20 GB
Total documents	816,863
Documents per source	100...13,625

TABLE I
CHARACTERISTICS OF INFORMATION SOURCES

IV. EXPERIMENTAL RESULTS

We have conducted a set of experiments to test the performance of our techniques. In these experiments, we used the beacon network to route keyword queries to information sources, and counted the total number of information sources contacted for each query. Our goal is to minimize the number of unnecessary sources contacted, so that we can reduce the load on sources, improve response time and enhance overall scalability. Here, we focus on (1) the effectiveness of our informed probing technique and, (2) the different architectures for routing. Extensive evaluation of the effectiveness of our ProbResults ranking function, including the quality of returned documents, is presented in [13].

A. Experimental setup

In our experiments, we used a beacon network to route queries among 2,500 Internet information sources. To ensure our experiments were repeatable, we created our own information sources on a cluster in our lab, and populated them with HTML documents downloaded from 2,500 .com, .net, .gov, .edu and .org websites. Sites were selected randomly from the top websites returned by the Google search engine for searches on ".com", ".net" etc. The proportion of websites of each suffix used matched the global proportion reported by Google. Each information source managed documents downloaded from one website, and processed keyword searches using the vector space model with TF/IDF weighting. The characteristics of our data set are shown in Table I. Some sources had many documents and some had few, just as in the actual Internet. We also ran other experiments (not reported here) where we varied the total number of sources in the network from 1,000 to 2,500. The results are consistent with those reported here. In fact, some queries are now easier to satisfy because there are more possible sources of information.

Our query set consisted of real WWW queries from the publicly available search.com query database. Because our data sources are a small subset of the whole internet, not all queries matched data, and we filtered out queries that did not match any results at any source. We then expanded the query-set into a Zipf-like distribution (with an exponent of 1) by duplicating randomly chosen queries; a Zipfian distribution is the observed distribution of several real query sets [35], [9], [22]. We also ran experiments with a non-skewed workload (the same workload, without the Zipf-like expansion). These results are not reported in detail, but show that the relative ordering of which techniques are best do not change, although all techniques perform better for a skewed workload.

For our experiments, we assume that users choose a random beacon. Real users may instead choose a beacon run by their organization or themselves. Our current prototype experimentally can support a peak rate of 144 user queries per second, running on a 2.8 GHz Xeon workstation. We assume that each user specifies a threshold T: the number of desired document results. This is similar to a search engine, where users usually only look at the first page or two of results. Users can request more results if the first results returned are not sufficient. Here, we used $T = 10$, although other experiments (omitted here) show that our results and techniques generalize to other values of T. As may be expected, larger values of T cause more sources and beacons to be contacted looking for results, in linear proportion to the increase in T.

Our beacon prototype is implemented in C++, and uses XML messages carried over HTTP to communicate between beacons. Also, a beacon accepts user queries and returns results via XML over HTTP, and queries information sources using HTTP. The cluster machines which ran the beacons had dual 550 MHz Pentium III Xeon CPUs and 4 GB RAM, running Linux. We were able to run 17 beacons per machine without difficulty. We configured beacons with an experience factor $EF = 10$ and constant $p_{min} = 0.0001$ (see Section II-A); we experimented with a range of values and found that these provided the best performance for many different scenarios.

B. Standalone performance of the informed probing technique

First, we examined the standalone performance of the informed probing technique. Each probe query consisted of 6-8 keywords extracted from the source's crawlable surface-web interface (as described in Section II-B).

Over all 2,500 sources, we were able to extract an average of 45 percent of each source's documents using a single informed probe. Table II show sample results for ten randomly chosen sources with different content sizes and types. These results show that informed probing is quite effective at warming the beacon cache with source content. Note that some sources responded to the informed probe with more results than others. The effectiveness

Data Source	*Total Size in documents*	*Percentage of docs returned*
AffiliateMatch.com	252	81.34%
showcase.netins.net	13625	16.91%
sourceforge.net	3277	20.01%
mineral.galleries.com	1400	25.35%
forbes.com	2108	14.32%
aausports.org	148	77.02%
dmoz.org	10339	19.42%
abceda.com	107	94.39%
aahn.org	101	86.13%
dallasobserver.com	3142	22.21%

TABLE II
RESULTS RETURNED BY A SINGLE INFORMED PROBE QUERY

of the informed probe depends on the information available on the HTML page from which the probe was extracted. If an informed probe does not extract much information from a given source, then the beacon will take longer to warm its cache for that source. For comparison, we also measured the number of results returned by 300 random dictionary word queries consisting of 6-8 keywords each; this probe size is recommended by [10], [19][2].Our experiments showed that, on an average, only 3 out of 300 random dictionary word queries returned results. This experiment demonstrated that much fewer informed probes are necessary to extract content from source compared to random probing.

Note that although informed probing retrieves a large number of documents, it does not extract a full summary of the source's content. Therefore, it is still important to continually augment the cache with results of user queries and experience weighting, as described in Section II. We report experimental results for combining the informed probing technique and ProbResults routing in Section IV-E.

C. Beacon network topologies in the flat architecture

Next, we examined the impact of the beacon network topology in the flat architecture. Recall that in this architecture, each beacon treats its neighbor beacons as regular sources, and produces a single ProbResults ranking of beacons and sources in order to route queries. We used a network of 100 beacons to route queries to 2,500 information sources. In general, we expect beacons to be assigned to 100 or so sources, but we felt that a network of only 25 beacons was too small for our experiments. A total of 50,000 queries were submitted to randomly chosen beacons.

We compared a random network of beacons to a spanning tree network of beacons. In the random network, each beacon had an average of five beacon neighbors. In the spanning tree, each beacon had up to four beacon neighbors. In both cases, the links between beacons were bi-directional.

The results are shown in Figure 5. As the figure shows, under both topologies, the performance of the network improves as the beacon caches warm up. However, the spanning tree topology achieves more overall efficiency than the random topology. After 50,000 queries the spanning tree beacon network only needs to contact 68 sources per query in order to find results, compared to 90 with the random topology (32 percent more than the spanning tree topology). With the random network, the warming of the beacon caches produces improvements in routing to information sources, but this improvement is reduced by the ineffective routing to other beacons. In contrast, with the spanning tree topology, beacons route queries to other beacons more effectively, and thus overall the routing improves as the beacon caches warm up. Changing the maximum degree of the spanning tree (e.g., doubling it) had minimal impact on these results.

The spanning tree topology is used with the flat architecture for the rest of the results reported in this paper. It may be possible to develop other alternative topologies that provide higher performance. However, in practice we have found the spanning tree topology to be quite effective.

D. Alternative routing architectures

We conducted an experiment to evaluate our various architectures for routing queries between beacons. We set up a network of 100 beacons to route queries to 2,500 information sources. We submitted 50,000 queries to randomly chosen beacons. Since this experiment is aimed at studying the effect of organizing beacons into different architectures and comparing their performance with a random walk based approach, we did not perform informed probing. The effect of informed probing on system performance is studied in the next section. We compared the *hierarchical*, *flat*, and *hybrid* architectures to a more traditional peer-to-peer architecture that used random walks to route queries among beacons. In the hybrid architecture, our setup consisted of 10 superbeacons and 90 leaf beacons with 9 leaf beacons connected to each superbeacon. For the random walk, beacons were organized into a network with a random topology (with an average of 5 neighbors), and random walking was used to route queries between beacons.

The results are shown in Figure 6. As the figure shows, the hierarchical, flat and hybrid architectures improve routing over random walking by more than a factor of two. The beacons are able to use their cached information and the ProbResults ranking to make better routing decisions than the random walk.

The best performance is achieved using the hierarchical architecture. The superbeacon sees every query, and collects a large amount of information about where to route queries. Load is distributed fairly evenly over the leaf beacons, although beacons with popular content receive more queries. The beacon network using the flat architecture contacts nearly 1.6 times as many sources compared to the hierarchical approach (after the caches are warm). Unlike the superbeacon, each beacon in the flat architecture only sees a fraction of the queries. As a result, beacons in the flat architecture cache fewer results than the superbeacon, and have less information for making routing decisions. Moreover, even when there is excellent routing information, queries must travel several hops in the flat network to reach the right beacon. This effect is illustrated by Figure 7, which shows the number of beacons contacted, per 1,000 queries, averaged over all 50,000 queries, with each technique. Each extra beacon visited potentially tries a few of its own sources before forwarding the query to the next beacon, increasing the total number of sources contacted. The result of these effects is that queries in the flat architecture must visit more beacons and sources. Also, the load distribution among beacons is more skewed than in the hierarchical case; beacons in the middle of the spanning tree must process and route more queries than those at the leaves. Despite this extra inefficiency, the

[2]Query probing sometimes uses a specialized dictionary, for example from a particular corpus. However, since we are dealing with websites from multiple domains, a domain dictionary is not appropriate and we used a general English dictionary, the Unix ispell dictionary.

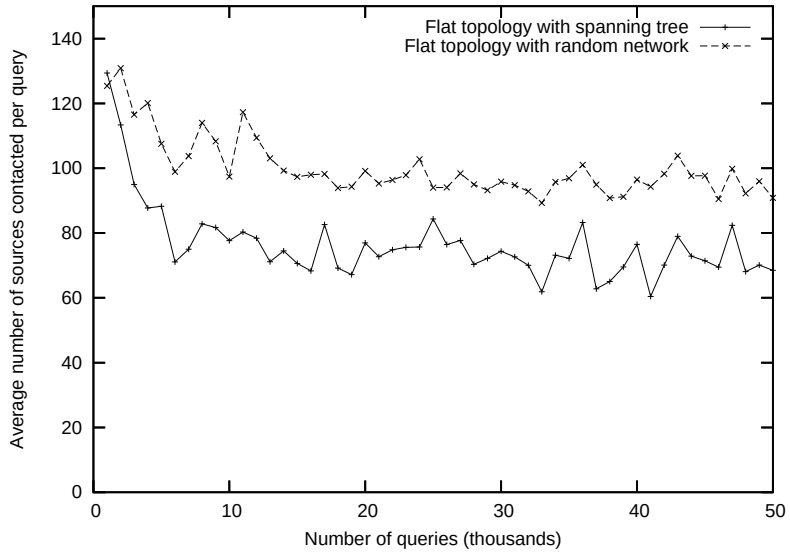

Fig. 5. Flat architecture topologies.

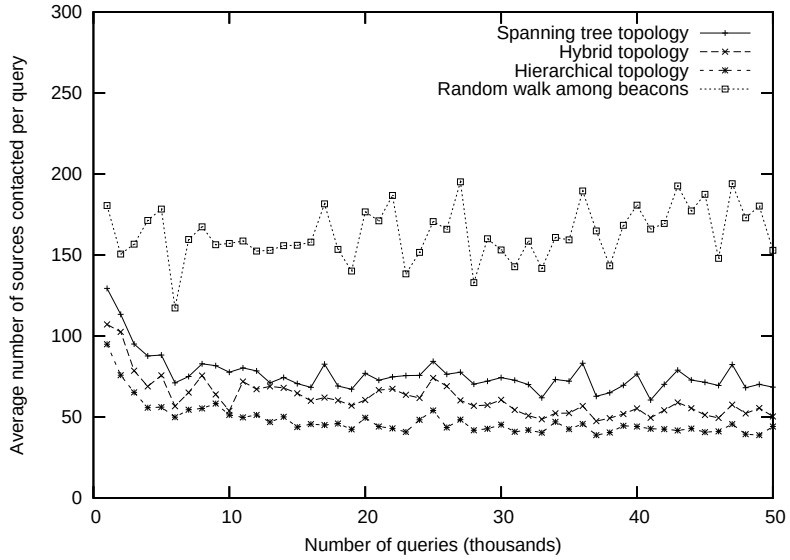

Fig. 6. Routing queries between beacons.

flat architecture may still be preferred since it avoids the potential bottlenecks of the centralized superbeacon approach.

In comparison, the hybrid network contacts, on an average, 20 percent fewer sources than the flat architecture. Beacons in the hybrid architecture contact only 1.1 times as many sources as those in the hierarchical architecture (after the caches are warm). The hybrid architecture also strikes a balance in the number of beacons contacted (as shown in Figure 7). The hybrid architecture, while preserving the decentralized structure of the system, is able to benefit from its closer resemblance to the hierarchical architecture, and can concentrate more routing information in the second level without having a central bottleneck.

We also experimented with increasing the number of sources per beacon. The result is that doubling the number of sources per beacon (from 10 to 20 in our experiments) results in a small bump in efficiency: 17 percent fewer sources contacted in the superbeacon topology and 19 percent fewer in the flat topology. As noted above, giving each beacon cache more direct information about sources allows beacons to make better routing decisions.

Other optimizations to our routing techniques may be possible. However, our results demonstrate the usefulness of ProbResults for routing queries between beacons. Moreover, the results demonstrate an interesting tradeoff between decentralization and source selection efficiency. If we can devote enough server resources to construct a superbeacon, the hierarchical approach is most effective. If not, we must choose the hybrid or possibly even the flat architecture, expending more system resources overall to avoid overburdening individual beacons.

E. Warming beacon caches with informed probing

Next, we examined the impact of using informed probing on the overall system performance. We compared the average number of sources contacted in an InfoBeacons system with and without

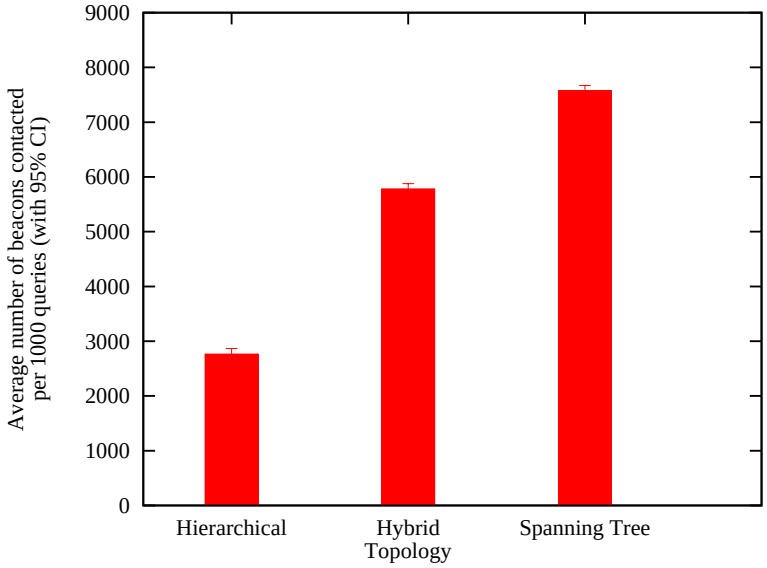

Fig. 7. Beacons contacted during routing.

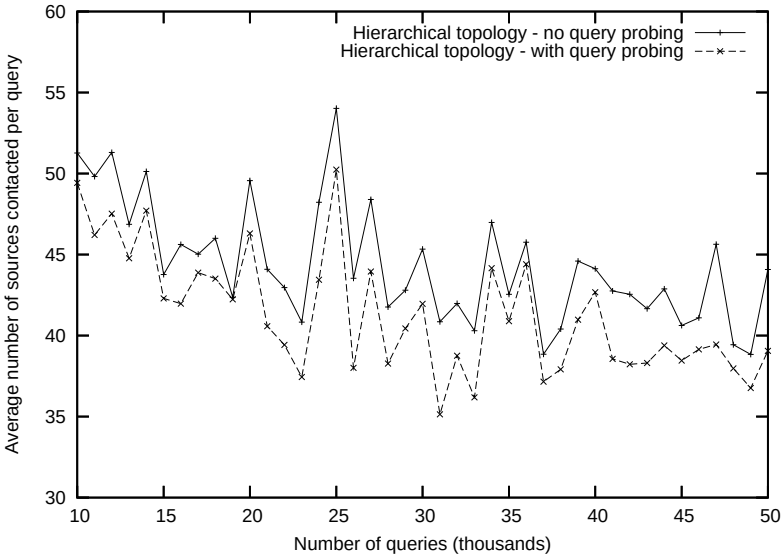

Fig. 8. Using informed probing with a hierarchical network.

informed probing. The beacons in this experiment were arranged in a hierarchical network. The probe queries were generated as described in Section IV-B and each source was queried with a single probe. Over all sources, the informed probes returned an average of 45 percent of a source's documents. Figure 8 shows that a system using informed probing contacted 10 percent fewer sources on average than a system with no probing. The main effect is that informed probing accelerates the warming of the cache, further improving the effective routing provided by ProbResults.

F. Result quality

Finally, we examined the quality of results returned by the InfoBeacons system. Because the data set is so large (over 800,000 documents), it is impossible to assign a relevance score by hand to each document for each query. Instead, we use a standard approach from information retrieval to compute relevance: each document is scored using the vector space model's cosine distance with TF/IDF weighting. This approach assigns a numeric score to each document for each query. We took the average relevance score for the documents returned for the flat and hierarchical approaches, as well as the random walk approach. In each case, the quality of results was roughly the same; although the random walk approach produced slightly better results, the relevance was only about 1 percent higher than the other approaches. Thus, the efficiency of the flat and hierarchical approaches does not cause a drop in document quality.

V. Related work

Several systems have been developed to perform information retrieval using a peer-to-peer architecture [30], [29], [33]. Beacons actually handle the "source selection" problem, while individual sources handle the "information retrieval" problem.

Networked information retrieval systems such as GlOSS [17] and CORI [15] perform source selection, and beacons are similar to these existing systems but adapted in several ways to work into a peer-to-peer architecture. As discussed in Section II, the key differences between our system and existing networked IR systems are: 1. the beacon is loosely-coupled to sources; 2. the InfoBeacons network is a decentralized collection of multiple beacons, rather than a central directory server; and 3. beacons continually adapt their cache and source scores based on the results of queries sent to sources. Ipeirotis et al [20] use a predictive model to determine when to update a source summary (e.g., beacon cache). Their technique could be considered an alternative to our approach of contacting the source on every query and continually updating the beacon cache.

Several peer-to-peer systems have been developed to perform source selection [5], [16]. These systems also expect sources to export content summaries to aid in routing. The Harvest system is an early example, with "brokers" that are similar to our beacons [8]. Harvest combines source data export with search engine-style crawling of static content by modules called "gatherers." Unlike Harvest, our system requires neither source export, nor that the data be crawlable (as much hidden-web data is not).

Callan, Lu and Renda have examined a peer-to-peer framework similar to InfoBeacons for searching multiple information sources [23], [28], [24], an approach they refer to as "federated search." Although InfoBeacons addresses some of the same problems and uses some of the same approaches as their work, there are several differences. First, InfoBeacons and federated search are useful for different types of application scenarios. Federated search deals mainly with a relatively static network of digital libraries. In contrast, InfoBeacons is designed to deal with data sources that appear or disappear frequently, and frequently update their content. Second, while [23], [28], [24] discuss one type of architecture (roughly analogous to our flat architecture), we have also examined the hierarchical and hybrid architectures, and compared and contrasted them with a flat architecture.[3] This previous work also deals with techniques for merging results, and such techniques could be used to augment the client-side merging performed by InfoBeacons. Balke et al [3] describe a peer-to-peer federated search system similar to InfoBeacons, in that it focuses on retrieving a subset of the results, and uses a spanning tree topology. Our work goes beyond the techniques of [3] in several ways, including: comparing hybrid and super-beacon topologies to the spanning tree topology; fuzzy matching between queries and cached results, instead of the exact query lookup of [3]; and dynamic adaptation of the beacon cache to progressively optimize the search performance.

Other systems that search multiple sources include data integration systems and search engines. Data integration systems, including traditional [12] and P2P systems [18], require tight schema integration. These systems construct complex schema mappings [6] or assume that most of the data has similar structure [18]. In a large scale system such as the Web, it is too expensive to construct all the required mappings, and data is structured in a wide variety of ways. Compared to these systems, our approach trades strong query semantics for enhanced scalability. Search engines [27] can search over HTML pages at many sites but do not deal well with uncrawlable or "hidden" data in web databases. Our approach uses sources' own query processors to search "hidden" data. Some search systems assume a consistent classification scheme or topic hierarchy to which sources can be assigned to aid in routing (such as in [19], [32]) but it is not clear that sources can always be assigned a single, unambiguous topic or that a single hierarchy is equally useful to all users.

Various approaches to routing in peer-to-peer systems have been proposed [36], [21], [26], [1], [31], [22]. Our system uses the full text of content to aid in routing, while these existing systems focus on document metadata, query statistics, network topology, or peer-processing capacity. It may be possible to combine our approach with existing approaches to achieve even more scalability and accuracy in routing.

Caching of data to improve performance has been well studied in many contexts, including the web [4], database systems [14], information retrieval [25] and peer-to-peer search [7]. Usually, data from a known source is cached to hide latency, not necessarily for source selection.

VI. Conclusions

We have examined how techniques adapted from information retrieval can be used to route queries in a peer-to-peer system. Our goal is to route queries to the best information sources, and allow those sources to perform the actual query processing. A network of beacons work together to perform the routing, by caching results from data sources and using this cache to route future queries. Our informed probing technique uses a small number of query probes to warm the cache more quickly than just caching previous query results. Next, we presented three approaches to inter-beacon routing. The hierarchical approach uses a single superbeacon to choose among beacons, who then choose among sources. In the flat approach, beacons treat other beacons in the network like data sources and assign ProbResults rankings to neighbor beacons and data sources. Finally, we examined the hybrid approach that retains the decentralized structure of the flat architecture while still gaining from the advantage of a hierarchical architecture. Experimental results demonstrate the performance/centralization tradeoff between the hierarchical, hybrid and flat architectures. These results also show that our techniques are more effective than random walks for routing queries. These results show the effectiveness of using information retrieval techniques for routing queries to information sources.

[3]Note that the terms "hierarchical" and "hybrid" are used differently in Callan, Lu and Renda's work. The papers [24], [28] refer to a "hierarchical" network of "hubs" and "leaf nodes;" such an organization is what we refer to as a "flat" network of "beacons" and "sources." The paper [23] refers to a "hybrid" network of hubs and leaf nodes, which is also similar to our "flat" network.

References

[1] L. Adamic, R. Lukose, A. Puniyani, and B. Huberman. Search in power-law networks. *Phys. Rev. E*, 64:46135–46143, 2001.

[2] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *Proc. SIGMOD*, 2003.

[3] W.-T. Balke, W. Nejdl, W. Siberski, and U. Thaden. Progressive distributed top-k retrieval in peer-to-peer networks. In *Proc. ICDE*, 2005.

[4] G. Barish and K. Obraczka. World wide web caching: Trends and techniques. *IEEE Communications Magazine*, May 2000.

[5] M. Bawa, R. J. Bayardo Jr., S. Rajagopalan, and E. Shekita. Make it fresh, make it quick — searching a network of personal webservers. In *Proc. WWW Conference*, 2003.

[6] P. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing: A vision. In *Proc. WebDB Workshop*, 2002.

[7] B. Bhattacharjee, Sudarshan Chawathe, Vijay Gopalakrishnan, Pete Keleher, and Bujor Silaghi. Efficient peer-to-peer searches using result-caching. In *Proc. Int'l Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[8] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz. The Harvest information discovery and access system. In *Proc. WWW Conference*, 1994.

[9] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *Proc. INFOCOM*, 1999.

[10] James P. Callan and Margaret E. Connell. Query-based sampling of text databases. *Information Systems*, 19(2):97–130, 2001.

[11] J.B. Caverlee, L. Liu, and D. Buttler. Probe, cluster, and discover: Focused extraction of qa-pagelets from the deep web. In *Proc. ICDE*, 2004.

[12] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *In Proc. of IPSJ (Information Processing Society of Japan) Conference*, October 1994.

[13] B. F. Cooper. Guiding queries to information sources with InfoBeacons. In *ACM/IFIP/USENIX 5th International Middleware Conference*, 2004.

[14] M.J. Franklin and M.J. Carey. Client-server caching revisited. In *Proc. Int'l Workshop on Distributed Object Management*, 1992.

[15] J.C. French, A.L. Powell, J. Callan, C.L. Viles, T. Emmitt, K.J. Prey, and Y. Mou. Comparing the performance of database selection algorithms. In *Proc. SIGIR*, 1999.

[16] L. Galanis, Y. Wang, S.R. Jeffrey, and D.J. DeWitt. Locating data sources in large distributed systems. In *Proc. VLDB*, 2003.

[17] L. Gravano, H. Garcia-Molina, and A. Tomasic. GlOSS: Text-source discovery over the internet. *ACM TODS*, 24(2):229–264, June 1999.

[18] R. Huebsch, J.M. Hellerstein, N. Lanham, B.T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *Proc. VLDB*, 2003.

[19] P. Ipeirotis and L. Gravano. Distributed search over the hidden web: Hierarchical database sampling and selection. In *Proc. VLDB*, 2002.

[20] P. Ipeirotis, A. Ntoulas, J. Cho, and L. Gravano. Modeling and managing content changes in text databases. In *Proc. ICDE*, 2005.

[21] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti. A local search mechanism for peer-to-peer networks. In *Proc. Int'l Conf. on Information and Knowledge Management (CIKM)*, 2002.

[22] B.T. Loo, J.M. Hellerstein, R. Huebsch, S. Shenker, and I. S toica. Enhancing P2P file-sharing with an Internet-scale query processor. In *Proc. VLDB*, 2004.

[23] J. Lu and J. Callan. Content-based retrieval in hybrid peer-to-peer networks. In *Proc. International Conference on Information and Knowledge Management (CIKM)*, 2003.

[24] J. Lu and J. Callan. Federated search of text-based digital libraries in hierarchical peer-to-peer networks. In *Proceedings of the 27th European Conference on Information Retrieval*, 2005.

[25] Z. Lu and K. S. McKinley. Partial collection replication versus caching for information retrieval systems. In *Proc. SIGIR*, 2000.

[26] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proc. of ACM Int'l Conf. on Supercomputing (ICS'02)*, June 2002.

[27] L. Page and S. Brin. The anatomy of a large-scale hypertext web search engine. In *Proc. WWW*, 1998.

[28] M.E. Renda and J. Callan. The robustness of content-based search in hierarchical peer to peer networks. In *Proc. of Int'l Conf. on Information and Knowledge Management (CIKM)*, 2004.

[29] P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *Proc. ACM/IFIP/USENIX International Middleware Conference*, 2003.

[30] S. Shi, G. Yang, D. Wang, J. Yu, S. Qu, and M. Chen. Making peer-to-peer keyword searching feasible using multilevel partitioning. In *Proc. International Workshop on Peer-to-Peer Systems*, 2004.

[31] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. SIGCOMM*, Aug. 2001.

[32] A. Sugiura and O. Etzioni. Query routing for web search engines: Architecture and experiments. In *Proc. WWW*, 2000.

[33] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proc. SIGCOMM*, 2003.

[34] J. Wang and F. Lochovsky. Data extraction and label assignment for web databases. In *Proc. WWW*, 2003.

[35] Y. Xie and D. O'Hallaron. Locality in search engine queries and its implications for caching. In *Proc. INFOCOM*, 2002.

[36] B. Yang and H. Garcia-Molina. Efficient search in peer-to-peer networks. In *Proc. Int'l Conf. on Distributed Computing Systems (ICDCS)*, July 2002.

[37] B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proc. ICDE*, 2003.