Using information retrieval techniques to route queries in an InfoBeacons network

Brian F. Cooper

College of Computing, Georgia Institute of Technology
cooperb@cc.gatech.edu

Abstract. We present the InfoBeacons system, in which a peer-to-peer network of beacons cooperates to route queries to the best information sources. The routing in our system uses techniques adapted from information retrieval. We examine routing at two levels. First, each beacon is assigned several sources and routes queries to those sources. Many sources are unwilling to provide more cooperation than simple searching, and we must adapt traditional information retrieval techniques to choose the best sources despite this lack of cooperation. Second, beacons route queries to other beacons using techniques similar to those for routing queries to sources. We examine alternative architectures for routing queries between beacons. Results of experiments using a beacon network to search 1,000 information sources demonstrates how our techniques can be used to efficiently route queries; for example, our techniques require contacting up to 70 percent fewer sources than random walk techniques.

1 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 [3]. 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, except that the beacons do not expect the

same level of cooperation from sources (i.e. exporting their indexes) that super-peers expect. In order to choose appropriate sources for a given query, we adapt techniques from networked information retrieval [14, 16, 13]. These techniques allow us to gather information about a source's content, and then predict how good that source will be for a given query. As a result, we can route queries through the system to the most appropriate sources, and avoid overburdening sources with irrelevant queries.

In this paper, we describe how the InfoBeacons system uses IR techniques to perform query routing. In particular, we describe routing at two levels. First, each beacon is responsible for several sources, and uses IR-style ranking to route a query to the best sources. A beacon cannot rely on sources to export a summary of their content, so the beacon builds up its own summary by caching the results of previous queries. The beacon then uses this cache to determine how to route future queries. Second, multiple beacons are connected in a peer-to-peer network, and IR-style ranking is used to determine how queries are routed through the beacon network. Many beacons can cooperate in this way to provide a system that searches a very large number of sources, while keeping the resource requirements low for each individual beacon. We examine two approaches to inter-beacon routing. In the *hierarchical* approach, a "superbeacon" uses IR-style ranking to choose among beacons, in the same way that beacons use ranking to choose among sources. In the *flat* approach, beacons treat each other as regular sources, forming a flat topology composed of both beacons and sources. A beacon routes queries to the most promising "neighbor," which may be a source or another beacon.

We have implemented an InfoBeacons prototype to route queries to information sources using our techniques. Experiments using our prototype to route queries to sources containing information downloaded from the Internet demonstrate that our techniques perform better than random walks, an efficient and scalable peer-to-peer routing mechanism [23, 1]. In ongoing work we are comparing our techniques to other routing approaches, such as those in [32, 20].

Content searching using information retrieval in peer-to-peer networks has been studied before [27, 26, 30]. Our work goes beyond these existing systems to examine using IR for routing between peers in addition to content searching. Moreover, our system focuses on searching over frequently-changing, uncooperative sources. Existing techniques, such as [26], assume that sources will export inverted lists for their content, and that these inverted lists do not require too many updates. General routing in peer-to-peer networks has also been studied extensively (for example, [32, 20, 23, 1, 28, 10, 21]). Some of these systems use a limited form of content-based routing based on document identifiers [28] or on keywords in document metadata (such as the title) [21]. Our work focuses on full-text content-based searching and routing, which presents new performance challenges. Other systems are not based on content, and instead route queries based on network topology [32, 23, 1] or peer processing capacity [10]. Routing approaches based on content, topology and capacity can be complementary, and it may be possible to combine these approaches. More related work is discussed in Section 5.

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

- We show how a beacon can route queries to appropriate sources using a ranking we have developed called *ProbResults.* (Section 2)
- We discuss how beacons can also use ProbResults to route queries to other beacons. We examine both *hierarchical* and *flat* approaches for inter-beacon routing. (Section 3)
- We present results from a preliminary experimental study comparing our routing techniques to existing techniques. Our study shows that ProbResults outperforms existing source selection techniques, and that both the hierarchical and flat approaches outperform random walk-based routing in our beacon network. (Section 4)

We examine related work in Section 5, and discuss our conclusions and future work in Section 6.

2 Routing queries to information sources

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 to sources, retrieve results, and return them to the user. The user is therefore shielded from the complexity of choosing and searching many different sources. A beacon is like a *meta-querier* such as GlOSS [16] or CORI [13], but adapted to work in a peer-to-peer manner with uncooperative sources. The differences between a beacon and existing meta-queriers are summarized in Section 5, and quantitative comparisons are presented in Section 4. User queries in our system are conjunctions of multiple terms, although our techniques can be extended to deal with general boolean queries.

In this section, we focus on the techniques a single beacon uses to route queries among its information sources. While each each beacon is only responsible for a few sources (say, 100 or so), the system scales to large numbers of sources by routing queries between multiple beacons (as described in Section 3). In this way, the resource requirements on each beacon are kept low. In fact, beacons are designed to be very lightweight peers, so that they can run in the background on a PC-class machine. Beacons can be run by users, data sources, libraries, ISPs, and so on.

It is too expensive to send every query to every source, so 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. As a result, a beacon must learn which sources are good for each queries, while relying only on the sources' basic search interface. We say in this case that the beacon is *loosely coupled* to the information sources.

Beacons learn about sources by caching results from previous queries, and then use these results to choose appropriate sources for future queries. The architecture that the beacon uses to carry out this process is shown in Figure 1. This figure shows four main components:

- The *user query interface*, which accepts queries from users and returns results.

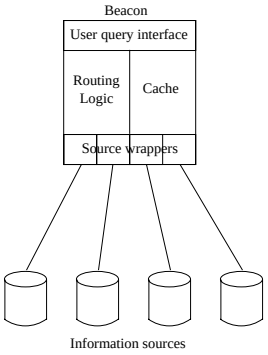

Fig. 1. Beacon architecture.

- The *source wrappers*, which submit queries to different information sources and retrieve results.
- The *cache*, which caches results from information sources.
- The *routing logic*, which uses the cache to choose which sources to route new queries to.

The source wrappers are lightweight: they only handle the task of connecting to the source, submitting a keyword query, and retrieving results. They do not perform complex schema translation or use advanced features of the source API. Techniques for generation of source wrappers have been studied by others; see for example [31]. This loose coupling ensures that it is cheap to integrate a new source, so that the system is tolerant to sources constantly appearing and disappearing.

We have developed a function, called *ProbResults*, to determine where to route queries. ProbResults uses the beacon 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 user specifies a desired number of results, and sources are contacted in order of decreasing ProbResults score until enough results have been found. (If the local sources cannot supply enough results, other beacons are contacted; see Section 3.) 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:

$$\mathit{ProbResultsScore}_Q^s = \prod_{i=1}^{n_Q} R_i^s / tq_s$$

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. We experimented with other ways of combining the R_i^s/tq_s

terms (such as addition or taking the max) but found that multiplication worked best because it gave a higher score to sources that return results containing all of the query terms than to sources that return results containing several instances of just some 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. To cache a new result document, the beacon parses the document, extracts the words, and increments the R_i^s values for each word for the source s. The result is that the beacon cache is very compact, and experiments show that a beacon responsible for 100 sources needs only a few tens of megabytes of cache. If necessary, the cache size can be bounded, resulting in graceful degradation in performance. Detailed results on the cache size are presented in [11].

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:

	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 [16]. 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. Results in Section 4 show that ProbResults produces better predictions in our system than the Ind ranking.

Two optimizations are useful to significantly improve the accuracy of the ProbResults function. First, if the beacon cache contains no information for a particular query

word i for a given source s, a non-zero minimum value P_{min} is used, instead of zero, for R_i^s/tq_s. This minimum probability addresses the fact that the cache is an incomplete summary of a source's content. In the example above, the $R_i^{s_1}$ value for "product" for s_1 might be zero because s_1 does not have any documents containing the term "product," or because the documents containing "product" have not yet been cached; the beacon does not know which case is correct. Without P_{min}, a source for which only some of the query words appeared in the cache would have a score of zero. Given a query for "product exothermic reactions," we still would prefer source s_1 over some source s_3 for which $R_i^{s_3}$ is zero for all three query terms. Using P_{min} instead of $0/100$ for "product" ensures that the ProbResults score for s_1 for this query is non-zero.

The second optimization, called *experience weighting*, weights the R_i^s values in the beacon cache to reflect the beacon's experience with word i as a query word. If a query containing word i is sent to source s and the source returns results, the R_i^s value is multiplied by a constant *experience factor* EF. If a query containing word i returns no results from s, then R_i^s is divided by EF. Experience weighting allows the beacon to refine its cache statistics based on the behavior of the source, and to make better predictions despite having incomplete information about the source's query model or content changes at the source.

Due to space limitations, both of these optimizations, as well as experimental results demonstrating their utility, are described elsewhere [11].

3 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" [23, 1] have been shown to be an effective and scalable way of routing queries in a peer-to-peer network. 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.

Beacons use the ProbResults ranking to route queries to information sources. We can extend this approach to use ProbResults to route queries between beacons. In particular, we study two 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.

An example of the *hierarchical* approach is shown in Figure 2(a). As the figure

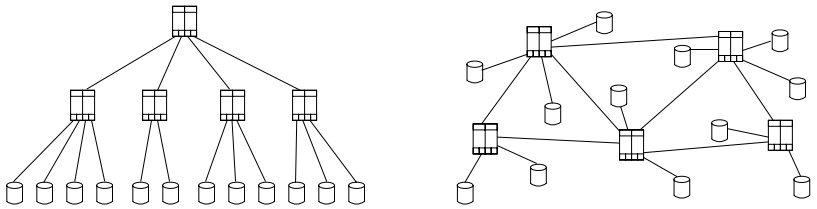

Fig. 2. Beacon network topologies: (a) hierarchical, (b) flat.

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 ProbResultsScore, 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 lightweight 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.

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.

A more scalable approach is to maintain the routing information in a decentralized manner, which is the goal of the *flat* architecture. An example of the flat architecture is shown in Figure 2(b). As this figure shows, a beacon's neighbors consist of both information sources and other beacons, forming a one-level "flat" topology. 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

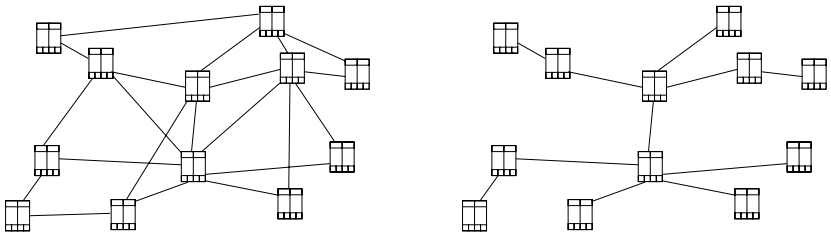

Fig. 3. Flat topologies: (a) random, (b) spanning tree. (Information sources omitted for clarity.)

disadvantage of the flat approach is that each beacon has less information than the superbeacon would, and thus prediction accuracy may suffer.

In experiments with our beacon prototype, we found that the topology 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 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 topology, as in Figure 3(b), the inter-beacon routing performs better. 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 4 demonstrate the performance improvement of the spanning tree topology.

It is possible that other peer-to-peer routing strategies might be used to route queries among beacons, including other strategies reported in the literature [32, 20]. As part of our ongoing work we are conducting experiments to compare these strategies to our ProbResults-based routing techniques.

4 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.

Our results can be summarized as follows:

- The ProbResults function is more effective than a random-walk routing strategy in routing queries to a beacon's information sources, reducing the number of sources

contacted per query by 90 percent. ProbResults is also more effective than routing strategies developed in non-peer-to-peer systems, such as GlOSS and CORI, reducing the number of contacted sources per query by at least 35 percent.

- Using a spanning tree topology to connect beacons in the *flat* architecture provides higher performance than a random graph topology, reducing the number of contacted sources per query by 23 percent.
- The *hierarchical* architecture provides the best performance overall; beacons in this architecture contact 70 percent fewer sources per query compared to beacons using a random walk strategy. The *flat* architecture, which avoids the potential bottlenecks of a centralized superbeacon, also provides good performance, with 31 percent fewer sources contacted per query compared to random walks.

We describe these results in more detail in this section.

4.1 Experimental setup

In our experiments, we used a beacon network to route queries among 1,000 Internet information sources. To ensure our experiments were repeatable, we created our own information sources on machines in our lab, and populated them with HTML documents downloaded from 1,000 .com, .net, .gov, .edu and .org websites. Each information source managed documents downloaded from one website, and processed keyword searches using the vector space model with TF/IDF weighting. The total number of documents at all sources was 166,145, for a total of 4.0 GB. Each source had between 1 and 2,303 documents. Some sources had many documents and some had few, just as in the actual Internet.

We used synthetically generated keyword queries so that we could evaluate our system with a large query set. The distribution of query terms in our generated queries matched the observed distribution of several real query sets as reported in [7], namely, that the most frequent query terms are terms that are neither too common nor too rare in the document corpus. Queries had between one and six terms.

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. Here, we used $T = 10$, although other experiments (omitted here) show that our results and techniques generalize to other values of 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.

4.2 Routing queries between information sources

First, we conducted an experiment to examine the performance of our techniques for routing queries between information sources. In this experiment, we used our beacon prototype to route queries between 100 sources selected randomly from our total set of 1,000. We used 40,000 queries generated from the terms in the content of these 100 sources. We compared several routing mechanisms:

- *ProbResults*: Our ranking function, described in Section 2.

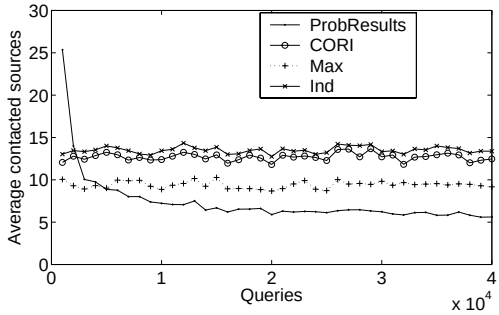

Fig. 4. Routing queries to information sources.

- *Ind*: The Ind ranking function, used in the bGlOSS system [16].
- *Max*: The Max ranking function, used in the vGlOSS system [16].
- *CORI*: The ranking function used in the CORI system [13].
- *Random*: Queries are routed to randomly selected sources.

The Ind, Max and CORI ranking functions are defined in [16, 13]. The GlOSS and CORI systems require that sources export their data to a central index to aid in routing, a requirement that is not feasible if sources are uncooperative. In such a case, query probing can be used instead: a set of randomly chosen queries are sent to the source and the results are used to substitute for the full source content [8]. This is the approach we use in our experiments.

The results are shown in Figure 4. The horizontal axis in this figure shows the number of queries submitted to the beacon, and the vertical axis shows the average number of sources contacted per query. Initially the beacon using ProbResults performs poorly, but as the cache warms up, the performance improves. Eventually (after about 5,000 queries), the beacon using ProbResults performs better than a beacon using the other techniques. With a warm cache, a beacon using ProbResults contacts 52 percent fewer sources than one using CORI, 35 percent fewer than one using Max, 55 percent fewer than one using Ind, and 90 percent fewer than one using Random (62.3 sources per query, not shown). ProbResults performs well because it effectively tracks the behavior of sources in response to queries. The warm-up time of the beacon cache is a disadvantage compared to existing techniques; in ongoing work, we are examining using query probing to accelerate the warming time.

We have also examined the quality of returned information, and experimented with smaller and larger numbers of sources per beacon, with sources that have larger content databases, and with sources that have frequently changing content. In each case, a beacon using ProbResults performs better than one using Ind, Max, CORI or Random. These results are discussed in detail in [11].

4.3 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

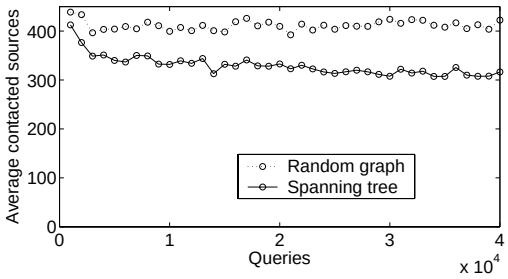

Fig. 5. Flat architecture topologies.

sources, and produces a single ProbResults ranking of beacons and sources in order to route queries. We used a network of 50 beacons to route queries to 1,000 information sources. In general, we expect beacons to be assigned to 100 or so sources, but we felt that a network of only ten beacons was too small for our experiments. In ongoing work we are conducting experiments with larger numbers of beacons and sources. A total of 40,000 queries generated from content at all sources were submitted to randomly chosen beacons.

We examined two topologies for the connections between beacons:

- *Random network*: connections were made between randomly chosen beacons to form a general graph. Each beacon had an average of 5 beacon neighbors.
- *Spanning tree*: connections were made between randomly chosen beacons to form a spanning tree. Each beacon had up to 4 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 improvement is most noticeable with the spanning tree topology. After about 20,000 queries the beacon network only needs to contact 316 sources per query in order to find results, compared to 412 with the random topology (30 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 cancelled 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.

The spanning tree topology is used with the flat architecture for the rest of the results reported in this paper. In ongoing work, we are examining other topologies and techniques to further optimize the flat routing architecture.

4.4 Routing queries between beacons

We conducted an experiment to examine the performance of our techniques for routing queries between beacons. We used a network of 50 beacons to route queries to 1,000 information sources. As before, we submitted 40,000 queries to randomly chosen beacons. We compared the following techniques:

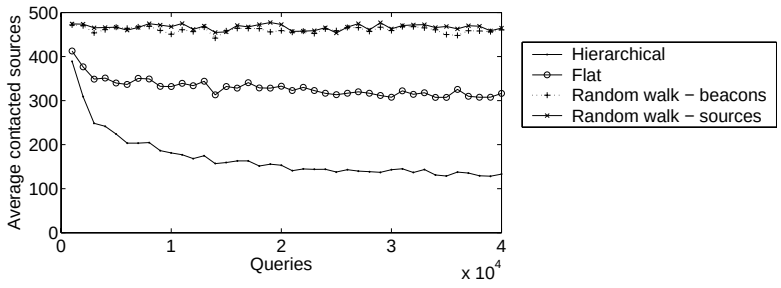

Fig. 6. Routing queries between beacons.

- *Hierarchical architecture*: A superbeacon used ProbResults to route queries to beacons, who then routed them to information sources.
- *Flat architecture*: Beacons used ProbResults to route queries to their neighbors in a flat topology of sources and beacons.
- *Random walk among beacons*: 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.
- *Random walk among sources*: Information sources were organized into a network with a random topology (with an average of 5 neighbors), and random walking was used to route queries between sources.

The two random walk approaches allowed us to examine different aspects of our techniques: *random walk among beacons* uses beacons but not IR-style routing between them, while *random walk among sources* uses a traditional unstructured peer-to-peer architecture instead of the beacon network.

The results are shown in Figure 6. As the figure shows, the best performance is achieved using the hierarchical architecture. The superbeacon, despite having limited information about the data accessible at each beacon, still has enough information to make informed routing decisions using the ProbResults metric.

The beacon network using the flat architecture contacts 2.3 times as many sources compared to the hierarchical approach (after the caches are warm). Unlike the superbeacon, which processes all 40,000 queries, 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. The result is less routing accuracy. The flat architecture may still be preferred, however, since it avoids the potential bottlenecks of the centralized superbeacon approach.

The flat architecture performs better than either random walk approach, contacting at least 31 percent fewer sources (after the caches are warmed). The beacons are able to use their cached information and the ProbResults ranking to make better routing decisions than the random walk. This demonstrates the value of information retrieval-style ranking to the query routing process.

Finally, the random walk among beacons approach performs only marginally better than the random walk among sources approach. This result demonstrates that the beacon

routing among sources is not by itself sufficient in a network of distributed beacons; user queries must also be carefully routed to the proper beacons.

In ongoing work, we are examining further optimizations to our routing techniques, and comparing our techniques to other peer-to-peer routing strategies (such as [32, 20]).

5 Related work

Several systems have been developed to perform information retrieval using a peer-to-peer architecture [27, 26, 30]. Beacons actually handle the "source selection" problem, while individual sources handle the "information retrieval" problem.

Meta-queriers such as GlOSS [16] and CORI [13] perform source selection, and beacons are similar to metaqueriers but adapted in several ways to work into a peer-to-peer architecture. First, the beacon is loosely coupled to the information source, while many existing systems require sources to export their contents or content summaries. This allows the beacons to work with sources that are not willing to provide more cooperation than simple searching. Loose coupling also makes it inexpensive to integrate a new source, addressing the high turnover observed in many peer-to-peer systems. Second, meta-queriers tend to be centralized, while in our system, sources are managed in a decentralized way by many beacons. Our approach enhances 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. Our results show that continual adaptation improves source selection performance (especially when content is changing frequently; see [11]).

Several peer-to-peer systems have been developed to perform source selection [3, 15]. These systems, like metaqueriers, 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 [6]. Harvest combines source data export with search engine-style crawling of static content through modules called "gatherers." Unlike Harvest, our system requires neither source export, nor that the data be crawlable (as much hidden-web data is not).

Other systems that search multiple sources include data integration systems and search engines. Data integration systems, including traditional [9] and P2P systems [18, 17, 24] require tight schema integration. These systems construct complex schema mappings [4] 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 [25] 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, 29]) 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 [32, 20, 23, 1, 28, 10, 21]. Our system uses the full text of content to aid in routing, while

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 accuracy in routing.

Caching of data to improve performance has been well studied in many contexts, including the web [2], database systems [12], information retrieval [22] and peer-to-peer search [5]. Usually, data from a known source is cached to hide latency, not necessarily for source selection.

6 Conclusions and future work

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 works together to perform the routing. In particular, we examined routing at two levels. First, individual beacons use ProbResults, a ranking adapted from information retrieval, to route queries to individual sources. Our techniques perform well despite limited cooperation from sources. Second, beacons also use ProbResults to route queries to other beacons. We presented two approaches for inter-beacon routing. In the hierarchical approach, a single superbeacon chooses among beacons, who then choose among sources. In the flat approach, beacons route queries to beacon neighbors if those neighbors have a higher ProbResults score than the beacon's sources. Experimental results demonstrate that our techniques are more effective than random walks for routing queries to information sources. These results show the value of using full-text content information when conducting peer-to-peer routing. In ongoing work, we are examining further routing optimizations, and comparing to other existing routing schemes.

References

1. L. Adamic, R. Lukose, A. Puniyani, and B. Huberman. Search in power-law networks. *Phys. Rev. E*, 64:46135–46143, 2001.
2. G. Barish and K. Obraczka. World wide web caching: Trends and techniques. *IEEE Communications Magazine*, May 2000.
3. 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*, 2003.
4. P.A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing: A vision. In *Proc. WebDB*, 2002.
5. B. Bhattacharjee. Efficient peer-to-peer searches using result-caching. In *Proc. IPTPS*, 2003.
6. C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz. The Harvest information discovery and access system. In *Proc. 2nd WWW Conference*, 1994.
7. B. Cahoon, K. S. McKinley, and Z. Lu. Evaluating the performance of distributed architectures for information retrieval using a variety of workloads. *ACM Transactions on Information Systems*, 18(1):1–43, January 2000.
8. J.P. Callan and M.E. Connell. Query-based sampling of text databases. *ACM TOIS*, 19(2):97–130, 2001.
9. 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 Conference*, October 1994.

10. Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making Gnutella-like P2P systems scalable. In *Proc. ACM SIGCOMM*, 2003.
11. B.F. Cooper. Guiding users to information sources with InfoBeacons. In *Proc. ACM/IFIP/USENIX 5th International Middleware Conference*, 2004.
12. M.J. Franklin and M.J. Carey. Client-server caching revisited. In *Proc. Int'l Workshop on Distributed Object Management*, 1992.
13. 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.
14. N. Fuhr. A decision-theoretic approach to database selection in networked IR. *ACM TOIS*, 17(3):229–249, 1999.
15. L. Galanis, Y. Wang, S.R. Jeffrey, and D.J. DeWitt. Locating data sources in large distributed systems. In *Proc. VLDB*, 2003.
16. L. Gravano, H. Garcia-Molina, and A. Tomasic. GlOSS: Text-source discovery over the internet. *ACM TODS*, 24(2):229–264, June 1999.
17. A.Y. Halevy, Z.G. Ives, P. Mork, and I. Tatarinov. Piazza: Data management infrastructure for semantic web applications. In *Proc. WWW*, 2003.
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. V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti. A local search mechanism for peer-to-peer networks. In *Proc. CIKM*, 2002.
21. B.T. Loo, R. Huebsch, I. Stoica, and J.M. Hellerstein. The case for a hybrid P2P search infrastructure. In *Proc. International Workshop on Peer-to-Peer Systems*, 2004.
22. Z. Lu and K. S. McKinley. Partial collection replication versus caching for information retrieval systems. In *Proc. SIGIR*, 2000.
23. 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.
24. W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Loser. Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks. In *Proc. WWW*, 2003.
25. L. Page and S. Brin. The anatomy of a large-scale hypertext web search engine. In *Proc. WWW*, 1998.
26. P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *Proc. ACM/IFIP/USENIX International Middleware Conference*, 2003.
27. 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.
28. 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.
29. A. Sugiura and O. Etzioni. Query routing for web search engines: Architecture and experiments. In *Proc. WWW*, 2000.
30. C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proc. SIGCOMM*, 2003.
31. J. Wang and F. Lochovsky. Data extraction and label assignment for web databases. In *Proc. WWW*, 2003.
32. 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.