Kosmix: High-Performance Topic Exploration using the Deep Web

Anand Rajaraman
Kosmix Corporation
Mountain View, CA, USA
anand@kosmix.com

ABSTRACT

Kosmix lies at the intersection of two important trends: topic exploration and the Deep Web. Topic exploration is a new approach to information discovery on the web that satisfies certain use cases not served well by conventional web search. The Deep Web, an inhospitable region for web crawlers, is emerging as a significant information resource. We describe the anatomy of Kosmix, the first general-purpose topic exploration engine to harness the Deep Web using a federated search approach. We focus in particular on the Kosmix approach to query tranformation and caching, which is essential to ensure reasonable performance.

1. INTRODUCTION

Web search engines, such as Google, Yahoo, and Bing, excel at finding the needle in a haystack: a single fact, a single definitive web page, or the answer to a specific question. Often, however, the user's objective is not to find a needle in a haystack, but to learn about, explore, or understand a broad topic. For example:

- A person diagnosed with diabetes wants to learn all about this disease. The objective is not just to read the conventional medical wisdom, which is a commodity available at hundreds of websites, but also to learn about the latest medical advances and alternative therapies, evaluate the relative efficacy of different treatment options, and connect with fellow-sufferers at patient support groups.
- A reporter researching a story on Hillary Clinton needs access to her biography, images, videos, news, opinions, voting record as a lawmaker, statements of financial assets, cartoons and other political satire.
- A traveler planning a trip to San Francisco needs to learn about attractions, hotels, restaurants, nightlife, suggested itineraries, what to pack and wear, and local events.

VLDB '09, August 24-28, 2009, Lyon, France

These are just three of numerous use cases where the goal is to explore a topic. Topic exploration today is a laborious and time-consuming task, usually involving several searches on conventional web search engines. The problem in many cases is knowing exactly what to search for; in the diabetes example above, if the diabetes sufferer knows there are patient-support groups, or that there might be alternative therapies they might be interested in, it's not hard to find them through conventional search engines. The bigger issue is that most people exploring a topic don't know what to look for.

Kosmix tackles this problem by creating a *topic page* for any topic. The goal of a topic page is to provide a 360-degree view of a topic. In each of the examples above, the Kosmix topic page includes all the information listed. In addition, the topic page also provides a list of related topics, conveniently grouped together, that suggest further areas to explore. The topic page uses a two-dimensional layout that is more reminiscent of a newspaper or a magazine than a search results page. This is not a coincidence. Magazines and newspapers, like topic pages, have been designed for browsing as opposed to search; the goal is to facilitate serendipitous information discovery. The need for topic exploration is validated by the fact that Kosmix and its sister property RightHealth today serve topic pages to over 10 million unique visitors every month.

A second distinction between Kosmix and search engines lies in the source of the information on the Kosmix topic page. Search engines construct their search results pages from their indexes, which in turn are built using web crawlers. A Kosmix topic page consists of a collection of modules. Each module is constructed using the result of an API call to a web service, done at the time of page construction. This allows Kosmix to tap into the Deep Web, the portion of the web not accessible to crawlers: social networking sites, media-sharing sites (photos, videos, documents), library catalogs, airline reservation systems, phone books, scientific databases and all kinds of information that lie concealed from view behind web forms. Some estimates have pegged the size of the Deep Web at upto 500 times larger than the Surface Web [16].

Although there are differences in the ranking algorithms used by different search engines, the fundamental architecture of web search engines is well-understood [2]. Of late, there has also been some interest in adding Deep Web data to search engines using an enhanced web crawler [9]. A few domain-specific vertical search engines, such as Mobissimo and Kayak, have adopted a federated search approach to the

Deep Web, accessing deep web sources at query-time and constructing results pages based on their responses. To our knowledge, the Kosmix explore engine is the first general-purpose, domain-independent service that uses a hybrid of the two approaches to harness the Deep Web.

The basic anatomy of the Kosmix explore engine is described in [12]. In this paper, we will focus on two challenges that arise in its implementation: query transformation and performance. We will provide a brief overview of the overall architecture of the explore engine in order to set the stage for understanding the query transformation and performance issues.

The rest of this paper is organized as follows. Section 2 describes the two approaches to harnessing the power of the Deep Web and evaluates their advantages and disadvantages, making the case for a hybrid approach. Section 3 describes the anatomy of the Kosmix explore engine, with emphasis on query transformation and performance. Section 4 describes related work. Section 5 concludes with some thoughts on the evolution of the Deep Web.

2. APPROACHES TO DEEP WEB SEARCH AND EXPLORATION

There are two fundamentally different approaches to incorporating the deep web into search or topic exploration engines.

- **Deep Web Crawl.** Crawl as much of the deep web as possible and incorporate it into a conventional search engine index.
- **Federated Search.** Use APIs to access deep web sources at query-time and construct results pages based on their responses.

Wright [16], using a fishing analogy, calls these approaches *trawling* and *angling* respectively, while Madhavan et al. [9] calls them *surfacing* and *virtual integration.* These approaches are also analogous to the warehousing and mediation approaches in data integration.

2.1 Deep Web Crawling

The crawl-based approach has the advantage that it fits well with the conventional web search model. The additional documents can be added to the search index and ranked using the existing ranking algorithm. The deep web crawl approach has been employed very effectively at Google, especially for tail queries [9].

As the web evolves from a broadcast medium to a participative medium, however, some drawbacks of the crawl based approach are becoming apparent.

User-generated content and social media. The Web 2.0 revolution has seen an explosion of sites dedicated to user-generated content (UGC). Examples include Wikipedia, YouTube, Flickr, and specialized sites in many subject areas such as TripAdvisor and Yelp. Such sites have removed friction from the content-creation process and lead to a sharp increase both in the number of contributors and in content of various media types. While search engines have strived to keep up with the deluge, the amount of information available on such services is growing faster than search engine index sizes. Social media sites such as Facebook, MySpace, and Twitter take this trend even further. The information on such sites is subject to various access controls. For example, users may permit only their "friends" to view certain information. In such cases, web crawlers may not have access to the information, since they don't work on behalf of a real user.

Real-Time. One of the weaknesses of the conventional web search architecture is the time lag introduced by the crawl and index process. While search engines have reduced this latency considerably over the past years, it is still a problem for information of a time-sensitive nature. Examples include ticketing and reservation systems of all kinds, auctions and shopping sites with limited product availability, and financial information related to the stock market.

One example that has captured the popular imagination of late is Twitter. When breaking-news events happen, Twitter users are often the first to post news and images online, often within seconds. Instances of this kind include the earthquake in China (May 2008) and the landing of US Airways 1549 on the Hudson River (February 2009). In such cases, there is a window of time where Twitter Search produces superior results over any conventional web search engine.

Specialized search engines. Search ranking algorithms such as PageRank evolved at a time when the web consisted primarily of hyperlinked HTML documents. However, a large fraction of useful content available today does not fit the old model and requires different ranking methodologies. For example, media-sharing sites such as YouTube and Flickr have access to information such as the reputation of the person who uploaded the content, number of views, and ratings. Another trend is the emergence of specialized search engines for specific tasks, such as Mobissimo and Kayak for travel; SimplyHired for job listings; Shopping.com and TheFind.com for products; and so on. As the web evolves from static documents to dynamic content repositories, it would seem that the most natural approach is to let such sites develop their own site-specific search engines, and then federate the results of these engines, or of domain-specific aggregators.

Information presentation. The results presentation model of web search engines has not evolved significantly in over 10 years. In the meantime, specialized services (either site-specific search engines, or category-specific aggregators) have developed innovative information presentation models for specific use cases. For example, Zillow for home prices; TheFind for products; Facebook for user profiles; and so on. Once again, a federated approach makes a lot of sense. Yahoo's SearchMonkey effort is a first step by search engines to address this issue. It allows content owners to submit a catalog of *rich snippets* for a set of queries. The limitation of this aproach is that the set of queries needs to be specified in advance, together with a static snippet for each query.

Availability of APIs. A growing trend is the evolution of websites into web services. Web services provide access to the contents and capabilities of the site via APIs. A handful of standards have evolved for such APIs, including REST and JSON. Crucially, we have reached a tipping point where the quantity of useful information available through such APIs is sufficient to build useful services.

Business model issues. The dominance of search engines as gateways to the web tends to commoditize web content and intermediate between content creators and consumers. Many content creators are therefore wary of allow-

ing search engines access to their entire data set. For example, Twitter does not provide search engines except Twitter Search access to its API. Facebook allows only limited access to search engines. Several newspapers have digitized archives that date back several decades, but do not make these archives available to search engine crawlers. Record labels do not make copyrighted music available for crawl. Such information can in many cases be accessed only through an API.

Infinite Wine in a Finite Bottle. Content created by algorithms is an interesting new trend.The beginnings of this trend can bee seen in the Wolfram Alpha, which can answer questions that are mathematical computations over data. Since the set of mathematical computations is infinite, trying to represent the Alpha as a finite set of web pages to fit in an index is a futile exercise. An API is the only sensible way to access such sites.

2.2 Federated Search

The dynamic query approach overcomes many of the limitations of the crawl-index-search approach outlined in Section 2.1. However, the approach comes with its own set of challenges.

Integrating APIs. Each web service comes with its own API, which uses different parameters from every other API. The results are also formatted differently. A piece of custom code, conventionally called a wrapper, is required to connect to each web service, limiting the scalability of the approach.

Source Selection. Given a topic and thousands of potential information sources, it is not practical to query every information source for each query. For example, it is not meaningful to send the query "diabetes" to TripAdvisor.com, a travel resource. If we indiscriminately query a source for topics irrelevant to it, we run into two issues. The first is the potential to clutter the results page with irrelevant information. The second is overloading web service providers, potentially bringing down their systems or getting them to reject future queries from the explore engine. Unlike in the case of an index-based search engine, source selection needs to be done without having access the content of the source.

Query Transformation. Kosmix users enter free form text queries, while the underlying information sources often support a richer query model. Once we deem an information source as potentially useful for a query, the next task is to rewrite the query in the manner most suited for that source. It should be noted that this problem is very different from that of rewriting structured queries, which has been explored in detail in the context of information integration [13, 8].

Results Layout. As we discussed in Section 2, many data sources present results using innovative methods tailored to the kind of data, and we would like to preserve this richness rather than degenerate to the lowest common denominator. In addition, since results are inherently of different types, it is unnatural to force a linear ranking across them.

Performance. Unlike a web search engine, a federated search engine needs to make external API calls in order to construct its pages. Therefore, we should expect this approach to be inherently slower than web search. Long response times can, however, have the effect of turning off users. Therefore, it is a huge implementation challenge to keep response times within acceptable limits.

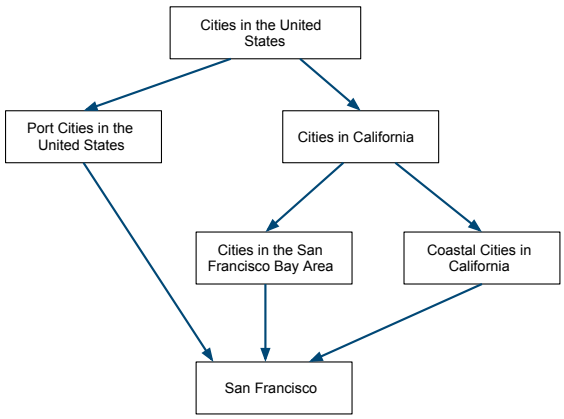

Figure 1: A small fragment of the Kosmix taxonomy

3. THE KOSMIX EXPLORE ENGINE

In this section we first provide a brief overview of the architecture of the Kosmix Explore Engine, followed by a detailed discussion of our approach to query rewriting and caching (to tackle the performance challenge described in the previous section). A companion paper [12] provides a detailed exposition of our approach to data access, data source selection, and results layout.

3.1 Architecture Overview

In the previous section, we listed some of the pros and cons of the crawl-based and federated search models. In practice, Kosmix uses a hybrid approach that combines features of the crawl and the federated search approaches. Some of the data is indexed locally, while API calls are made in other cases. Certain criteria are used to determine whether to locally index the data from a data source, or to query it using its API. These criteria are laid out in [12]. These criteria relate to the drawbacks of the crawl and index approach desribed in the foregoing section.

The cornerstone of the Kosmix explore engine is its taxonomy and categorization technology. The Kosmix taxonomy consists of millions of topics organized hierarchically, reflecting is-a relationships. For example, San Francisco is-a city. The resulting hierarchical structure is a directed acyclic graph (DAG). Figure 1 shows a small piece of the Kosmix taxonomy. Details on how the taxonomy is constructed and the challenges we faced are in [12].

The second key to the Kosmix explore engine is the Kosmix Categorization Service (KCS). Given a user query, KCS determines the nodes in the taxonomy that are most closely connected with the query. Let us say the query is "Pinot Noir." KCS determines that Pinot Noir is a kind of wine, which is a related to foods and beverages. It also determines that Pinot Noir is a kind of wine grape, and is related to viticulture and vineyards. Figure 2 shows a small selection of the full list of topics KCS determines are related to Pinot Noir.

In addition, KCS also determines the data sources in the system that are most likely to have data relevant to the query. Continuing with the Pinot Noir example, sites deemed relevant to this topic include Epicurious and Food Network (food and recipe-related websites), DailyPlate and FatSe-

Figure 2: Topics related to Pinot Noir

cret (both databases listing nutritional value of food), and Amazon.com (for wine shopping). In addition, there are also many general-purpose services that have content on a wide variety of topics, including Wikipedia, Google Image Search, and YouTube.

The next step is to query each of the selected sources and gather results. The user query may need to be transformed before sending it to certain data sources, as described in Section 3.2. The results so gathered are not laid out in a linear fashion, but grouped together by information type into a 2-dimensional layout: text, audio, video, user profiles, shopping, conversations and so on. The page real estate allotted to a group varies based on various factors, such as the relevance scores of the data sources in the group. Within a group, each data source gets a variable amount of real estate depending upon its relevance and other factors.

The *Related in the Kosmos* module enables exploration by surfacing topics in the taxonomy that are related to the query, grouped in a fashion that makes it easy to to scan. Figure 2 shows some of the related topics for the query "Pinot Noir." Clicking on one of these links takes the user to the page for that topic.

3.2 Query Rewriting

The general problem of reformulating a query to conform to an arbitrary API is a very difficult one. It is related to the work on query transformation done in the context of data integration [8, 13]. In the context of the explore engine, we made the observation that most of the data sources of interest support freeform text queries. We made a conscious design choice to restrict ourselves to such sources.

In practice, this is not a huge restriction. Topic exploration queries tend to be simpler and shorter than web search queries. For example, "flights from seattle to boston" is not a common use case for topic exploration. There are some exceptions, such as location inputs (e.g., zipcodes) that can be handled as special cases.

There are one scenario where query rewriting makes a big difference: *ambiguous queries.* These are queries such as "jaguar", which might mean the animal, the car, or the football team, among its many different meanings. When a user enters an ambiguous query, the explore engine presents choice of meanings: e.g., Jaguar (Animal), Jaguar (Car), Jacksonville Jaguars. The user can pick one of these meanings to get the topic page for the corresponding topic.

The question is, how to send such queries to the selected data sources? Sending the query in its unmodified form will likely produce many irrelevant results. There are two approaches the explore engine uses in such cases:

- **Textual Disambiguation.** Add keywords to the original query so as to convery the desired meaning. For example, add the keyword "car" to the original query "jaguar" to convey that the user meant that sense of the term.
- **Category Filter.** Many web services, especially those with large amounts of data such as YouTube and Flickr, support a search option that allows both a freeform text query and a category input. A category is usually a broad subject area, such as "health", "travel", or "science." The query is interpreted as: Find results containing the specified category, and also belong to the specified category. For example, the query "jaguar category:auto" on YouTube produces very different results than the standalone query "jaguar."

Each data source selected for the query at hand falls in one of three buckets:

1. The data source is known to contain data only for certain subject areas in the taxonomy, and the subject areas covered by the data source admit of only the user-selected meaning of the query term. For example, Vast.com is a service for listing used cars for sale. The term "jaguar" has precisely one meaning for this data source.
2. The data source covers a range of subjects that admits multiple meanings for the original user query, and supports a category filter. e.g., YouTube.
3. The data source covers a range of subjects that admits multiple meanings for the original user query, but does not support a category filter.

In the case of Type 1 data sources, we can send the user query without any modifications. For Type 3 data sources, the explore engine uses textual disambiguation. If a data source supports a category input (Type 2), that method is usually preferable to textual disambigaution. This is because most textual disambiguation is lossy: it forces the data source to find data items that mention more keywords than the original query, and therefore hurts recall. A properly implemented category filter does not have this problem. In practice, many data sources do not implement their category filters in the best possible manner, but the general point still holds.

For sources that support a category filter, the challenge is to pick the right category input. The explore engine uses the query categorization provided by KCS to find the closest category input from a menu of known choices for each data source.

3.3 Caching

It should be apparent from the foregoing discussion that building a topic page requires many calls to external services outside the control of the explore engine. In a naive implementation, that could lead to unacceptable response times

for users. As traffic to the explore engine scales, it could also place an unacceptable load on data sources.

The solution we have adopted at Kosmix is an intelligent caching layer. The simplest approach would be to cache topic pages. This, however, is not ideal, because different data types have different freshness requirements, e.g., news, Twitter and TripAdvisor hotel reviews. Therefore, we cache the results of each request to a data source, and assemble the topic page on the fly from these. The cache key therefore is a pair (data source, query).

Each cache entry has two different expiry times: a *soft expiry* and a *hard expiry.* When a query hits a soft-expired cache entry, the data in the cache may still be used to satisfy the query, but the (data source, query) pair is added to a background cache priming queue. When a query hits a hard expired cache entry, a call must be made to the external source on the fly to construct the topic page, and the cache entry gets refreshed as a by-product. A continuously running backgound priming process runs through entries in the cache priming queue, refreshing soft-expired cache entries in order of popularity.

Consider a query being processed by the explore engine. The data source selection process typically identifies 20-30 data sources for every query. Thus, a single query results in 20-30 cache lookups. We clearly cannot store the entire cache in memory (it is of the order of several terabytes of data, and grows constantly). Therefore, each cache lookup results in a random access to disk. For the SATA-2 disks that are in common use, the seek time is around 10 milliseconds, and so a cache lookup might take over half a second.

Half a second is clearly unacceptable for cache lookups. Fortunately, a new technology that is becoming popular comes to our rescue: *solid-state disks.* A solid-state disk (SSD) [15] uses solid-state memory (such as flash) to store persistent data. The random access time for the solid-state disks we use is 100 microseconds, which is two orders of magnitude faster than conventional disks. Thus, cache lookup for a query takes no more than a few milliseconds. In addition to much faster random seeks, SSDs also support many more I/O operations per second (IOPS) than hard disks, thus leading to better throughput under concurrent query loads. The price-point of SSDs is falling rapidly and is at a point where it is practical for us to use in our cache.

Kosmix uses a distributed cache, where each cache node runs a modified verion of ehCache [5], a widely-used open source cache. We have modified ehCache extensively, optimizing it both for SSD-based storage as well as large-scale data (in the order of several terabytes). Each cache entry is replicated, so data is not lost when a cache node fails. Kosmix plans to open-source our modifications to ehCache in the near future.

The cache allows us to render topic pages with acceptable latencies. Pages where most of the data is in the cache render in under a second, while most other pages render with a a few seconds. That definitely is slower than modern web search engines, but users appear to understand that the freshness of the data is often worth the additional wait.

4. RELATED WORK

Broder [3] and several other works have classified search queries into three categories: *navigational*, *informational*, and *transactional*. While the exact proportions have varied, there is agreement that a large fraction of queries are informational. Topic exploration is a good metaphor for informational queries.

There is a significant body of work on creating vertical search engines for specific domains by constructing semantic mappings from a mediated schema (or form) to collections of forms within a domain [4, 8, 14, 18]. Most of this work assumes that the mediated schema can be created by hand. Our work, on the other hand, focuses on a general-purpose, domain-independent explore engine. The sheer breadth of queries we need to handle makes it impossible to manually create a mediated schema, leading us to the automated creation and maintenance of a taxonomy.

Google's approach to crawling the Deep Web is described in [9]. There has been prior work around acquiring documents from databases with restricted query interfaces, as well as computing keyword distributions that summarize database contents to facilitate source selection [1, 6, 7, 10, 11, 17]. There has been significant research on query transformation and rewriting in the context of data integration [8, 13].

5. CONCLUSION

We described the anatomy of Kosmix, the first general purpose Deep Web Explore Engine. Kosmix enables a new approach to information finding, called topic exploration, using a hybrid approach to the Deep Web that combines elements of the crawl and federated search approaches. Traffic to Kosmix and its specialized health property RightHealth continue to increase at a rapid clip, demonstrating the value of the Kosmix approach to deep web exploration.

The architecture of web search engines today reflects the historical dominance of the Surface Web. The increasing importance of the Deep Web is forcing a rethink of this architecture. The hybrid approach to the Deep Web has several advantages over both the crawl-only and pure federated approaches. Some standardization around web service APIs could dramatically increase the scalability of this approach.

Acknowledgements

The Kosmix explore engine described in this paper is the collective effort of the Kosmix team. Special thanks to Tom Macke, Guy Albertelli, and Bruce Wright for help with the section on caching using SSDs.

6. REFERENCES

[1] L. Barbosa and J. Freire. Siphoning hidden-web data through keyword-based interfaces. In *SBBD*, 2004.

[2] S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *WWW7*, 1998.

[3] A. Broder. A taxonomy of web search. In *SIGIR Forum*, 36(2):3-10, 2002.

[4] A. Doan, P. Domingos, A. Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *SIGMOD*, 2001.

[5] Ehcache. `http://ehcache.sourceforge.net`.

[6] L. Gravano, P. G. Iperiotis, M. Sahami. QProber: A system for automatic classification of deep-web databases. *ACM Transactions on Information Systems*, 21(1):1-41, 2003.

[7] P. G. Iperiotis and L. Gravano. Distributed Search over the Hidden Web: Hierarchical Database Sampling and Selection. In *VLDB*, 2002.

[8] A.Y. Levy, A. Rajaraman, J.J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *VLDB*, 1996.

[9] J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, A. Y. Halevy. Google's Deep Web crawl. *PVLDB* 1(2): 1241-1252, 2008.

[10] A. Ntoulas, P. Zerfos, and J. Cho. Downloading textual hidden-web content through keyword queries. In *JCDL*, 2005.

[11] S. Raghavan and H. Garcia-Molina. Crawling the Hidden Web. In *VLDB*, 2001.

[12] A. Rajaraman. Kosmix: Exploring the Deep Web using Taxonomies and Categorization. In *IEEE Data Engineering Bulletin*, June 2009.

[13] A. Rajaraman, Y. Sagiv, and J.D. Ullman. Answering Queries using Templates with Binding Patterns. In *PODS*, 1995.

[14] J. Wang, J.-R. Wen, F. Lochovsky, and W.-Y. Ma. Instance-based schema matching for web databases by domain-specific query probing. In *VLDB*, 2004.

[15] Wikipedia. Solid-state Drive. `http://en.wikipedia.org/wiki/Solid-state_drive`.

[16] A. Wright. Searching the Deep Web. In *CACM*, 51(10):14-15, October 2008.

[17] P. Wu, J.-R. Wen, H. Liu, and W.-Y. Ma. Query selection techniques for efficient crawling of structured web sources. In *ICDE*, 2006.

[18] W. Wu, C. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the Deep Web. In *SIGMOD*, 2004.