Web Search Engines: Practice and Experience

Tao Yang, University of California at Santa Barbara
Apostolos Gerasoulis, Rutgers University

1 Introduction

The existence of an abundance of dynamic and heterogeneous information on the Web has offered many new opportunities for users to advance their knowledge discovery. As the amount of information on the Web has increased substantially in the past decade, it is difficult for users to find information through a simple sequential inspection of web pages or recall previously accessed URLs. Consequently, the service from a search engine becomes indispensable for users to navigate around the Web in an effective manner.

General web search engines provide the most popular way to identify relevant information on the web by taking a horizontal and exhaustive view of the world. Vertical search engines focus on specific segments of content and offer great benefits for finding information on selected topics. Search engine technology incorporates interdisciplinary concepts and techniques from multiple fields in computer science, which include computer systems, information retrieval, machine learning, and computer linguistics. Understanding the computational basis of these technologies is important as search engine technologies have changed the way our society seeks out and interacts with information.

This chapter gives an overview of search techniques to find information from on the Web relevant to a user query. It describes how billions of web pages are crawled, processed, and ranked, and our experiences in building Ask.com's search engine with over 100 million north American users. Section 2 is an overview of search engine components and historical perspective of search engine development. Section 3 describes crawling and offline processing techniques. Section 4 describes the online architecture and query processing techniques. Section 5 explains ranking signals and algorithms. Section 6 discusses the metrics for evaluating a search engine. Finally, Section 7 concludes this chapter.

2 Search Engine Components and Historical Perspective

In general, a search engine consists of three main components as shown in Figure 1: a crawler, an offline processing system to accumulate data and produce searchable index, and an online engine for realtime query handling. Their roles are summarized as follows.

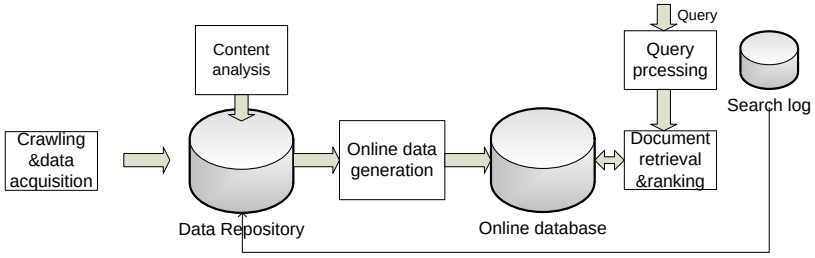

Figure 1: Components of web search.

- **Crawling and data acquisition.** A crawler discovers and adds the content of the Web to the search engine's data repository. Type of content gathered includes text pages, images, videos, and other types of content. Most crawlers find information by beginning at one page and then follow the outgoing URL links on that page. Therefore, if a page is not linked to from another page, this page may never be found by a search engine. In some cases, a search engine may acquire content through private data feeds from other companies. All of the information that a web crawler retrieves is stored in a data repository, which provides a foundation to build an online index searchable by users.

- **The data repository and offline processing.** Some search engines have extremely large databases with billions of pages while others have comparatively small ones. A search engine may aggregate results from its own database and from other search engines hosted in different locations to expand the database coverage. Offline processing mainly takes a collection of pages and builds appropriate data structure for search. The offline system also performs duplicate and spam detection and conducts additional data analysis to identify properties of a page as ranking signals.
- **Online search engine and ranking.** A search engine runs as a web service to answer queries from users. The query processing system uses the index to find relevant documents and rank them. It generates a result page which contains top ranked documents and serves this page to a user. As users interact with the search results, a search engine logs their browsing behavior and leverages such data for improving search quality.

Information retrieval researchers have studied document search for many years [64] before web search became popular in earlier nineties. An earlier system for locating Internet content was Gopher, used for file name lookup. One of the first "full text" crawler-based search engines was WebCrawler, which came out in 1994. As the World Wide Web became popular, a number of search engines and portals offered search services, including AltaVista, Excite, Infoseek, Lycos, Inktomi, Northern Light, Yahoo!, and AOL.

In earlier 1999, Google's search engine started to gain popularity with its new link-based ranking paradigm and a simple user interface. In 2002 and 2003, Yahoo! acquired Inktomi search and Overture which owned AltaVista and AlltheWeb, building its own search technology. Ask.com (formally called Ask Jeeves) acquired Teoma search in 2001 for providing search and question answering. Microsoft initially used search results from Inktomi and has gradually built its own engine: MSN search, Windows Live Search, Live Search, and Bing.

In Europe, Fast search was launched in 1999 to compete with Google and was acquired partially by Overture in 2003 and partially by Microsoft in 2007. In China, Baidu was started in 1999 and has become a dominating search engine. In Russia, Yandex search launched in 1998 and is currently dominating while in South Korea, Naver is a popular search engine launched in 2000.

Technology behind the major search engines has been seldom published because of fierce competition while some of their techniques have gradually appeared in patents and academic conferences. Academic conferences that cover more on search algorithms and data analysis include SIGIR, WWW, WSDM, and ECIR. Other related conferences for web data analysis include KDD, CIKM, SIGMOD, and VLDB. The system technology covering large-scale Internet systems and data platforms has appeared in various system conferences such as OSDI. Search engines have a major traffic impact to web sites, and the search industry activity is closely watched by web masters and various organizations including searchenginewatch.com. Recent text books that describe search algorithms and mining techniques can be found in [53, 4, 22, 11].

There exists a number of open-source search engines and related data processing tools available for search engine research and applications. The Apache Lucene/Solr package [50] is a Java-based search engine in which Lucene provides data indexing and full-text search capability while Solr provides a search user interface built on top of Lucene with additional support such as caching and replication management. Lemur [47] and Xapian [77] are open-source search engines with C++. Various open source code is available for underlying system support. Hadoop's MapReduce [33] is a popular clustering package for parallel processing of offline data. Apache HBase [34] and Cassandra [12] provide the key-value data store support for accessing large data sets.

3 Crawling and Data Processing

3.1 Crawling

A web crawler is part of the search engine to gather data from the Internet; it can recognize and collect HTML pages and other types of documents including PDF, PowerPoint, Word, and Excel. It extracts text information from these documents through conversion software so that corresponding text information is indexable and searchable. Figure 2 (a) illustrates the architecture of a large-scale crawler we have developed in the past for discovering and downloading billions of web pages.

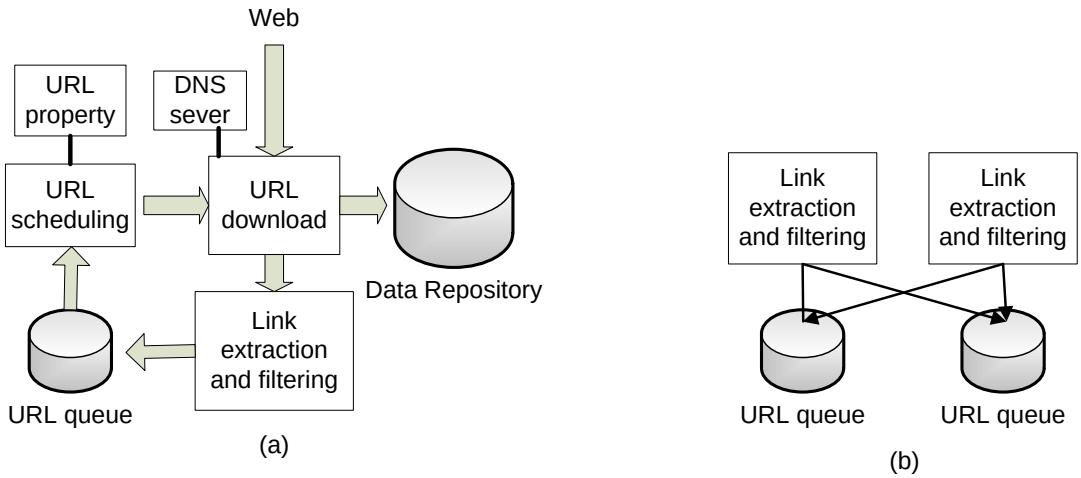

Figure 2: (a) Architecture of a web crawler. (b) Collaborative outgoing link extraction and URL distribution.

- The URL downloader fetches web pages following the HTTP protocol. Typically, a crawler uses a thread to handle an HTTP connection and maintains many concurrent threads to saturate the available download bandwidth. Since there can be a high latency in downloading a page from a site, a crawler machine can have a large number of HTTP requests outstanding. That increases system resource requirement as each machine needs to maintain information on all open connections.

- A DNS server resolves the IP address of URLs. Since there is a massive amount of DNS lookup of URL hosts, extra caching support can be useful.

- The URL extraction module parses downloaded pages and identifies the outgoing links. It may filter some of those URLs identified as low quality or duplicates. After filtering, it adds the extracted outgoing URLs to a URL queue. Since a crawler typically runs its service on a cluster of machines, extracted URLs from one machine may be distributed to URL queues in multiple machines as illustrated in Figure 2(b).

- The URL scheduling decides which URLs are to be visited in the next round of crawling. Given that web sites have variable sizes, small sites can be crawled quickly while it takes a long time to visit a large site. A URL scheduler periodically analyzes the properties of the queued URLs, prioritizes them, and selects a subset of URLs to the URL download process, to accomplish the following goals: 1) large sites are crawled completely; 2) URLs are recrawled frequently to ensure freshness; 3) revisiting for recently crawled URLs are excluded to avoid the endless revisit of the same URLs; and 4) low-quality or duplicate pages are not crawled or crawled with a low priority.

- The URL property service maintains properties of URLs. There is a resource contention to accomplish these goals in parallel, and the URL scheduler uses properties of URLs such as the last successful crawling time and unsuccessful data fetch attempts to make a sound decision. Since there is a large number of data accesses in parallel at each second on each crawling machine and there are a large number of URLs to be crawled, accessing URL property information can become a crawling bottleneck. The design of an efficient data structure and crawler architecture is therefore crucial.

There are a number of factors that further complicate crawler management.

- **Politeness.** A web site often hosts a large number of pages. To fetch these pages quickly, a crawler can occupy a significant amount of computing and bandwidth resource from this site, and this can affect the normal operation of such a site. To mitigate the negative impact on the site's resources, a crawler must follow a politeness policy to limit its crawl rate. For instance, a crawler may not make more than one request to the same host at the same time, and possibly it should add a time delay between two consecutive requests to the same host.

- **Robots exclusion protocol.** A web administrator can exclude their pages from the index of a search engine by providing the /robots.txt file in the top directory of the web site. This file specifies the visit restriction for a search engine's crawler [63]. For example, a line with text "Disallow: /tmp/" in example.com/robots.txt means that visiting example.com/tmp is not allowed. When fetching a page from a site, a crawler must check the robots.txt from the corresponding web host. To avoid repeated access of the robot.txt file from the same host, a crawler may save a copy of such a file in its local cache system with a periodical update.

- **Crawling abnormality.** Some sites respond to a crawler's request slowly, or even fail to respond. Some sites may return wrong or broken results to a crawler. A crawler needs to be resilient to the failure or slowness of a web site and be able to retry at a later time.

  Another type of crawling abnormality is called *crawling traps*. In this case, a web site keeps a crawler busy by dynamically feeding an infinite number of useless web pages. For example, there can be an infinite number of calendar pages generated dynamically and accessible through "Next day" button in a site. Such a trap can be detectable by site-oriented similar content analysis.

- **Hidden web and sitemap.** Crawlers from major search engines can effectively collect surface-level web pages, but there is a vast amount of hidden deep web information invisible to search engines. Any page that is not explicitly pointed by other web pages cannot be found by a crawler through hyperlink analysis. Dynamically-generated pages that require HTML-form or Javascript submission are often invisible. Automatic form filling can explore deep web [51], and there is still a challenge to reach high accuracy or coverage. An alternative method is to use the Sitemaps protocol [69] which enables webmasters to inform search engines about a list of pages that are available for crawling from their sites. A Sitemap is an XML file that lists URLs for a site along with additional metadata about each URL. Thus webmasters can advertise deep URLs which are not even hyperlinked from the surface web.

The freshness and content coverage of a data collection crawled are important for the quality of a search engine. The challenge a crawler faces is to discover newly created or updated pages on the web quickly. For known URLs, a crawler can revisit them frequently to detect if their content has been changed. The historic change frequency of a page and its site can guide how often a crawler should revisit this page. While crawling sites frequently can discover new pages, leveraging information published by a site such as RSS feeds can speedup this discovery process. An RSS feed is an XML-based file that provides a structured list of new pages added to a site. The RSS feeds are commonly associated with blogs and other information publishers. For example, CNN provides news RSS feeds under different categories. Amazon publishes RSS feeds on popular products with tags. There are additional signals that can be mined from the web to facilitate the discovery of new URLs, especially for hot or new topics. This includes the detection of new URLs that appear in news, blogs, search logs, and social network sites.

Earlier work that addresses parallel crawling and freshness appears in [17, 18]. The order of crawling to fetch important pages under resource constraints was studied in [3]. A research crawler and related techniques were discussed in [46]. Focused crawling was addressed in [14] with content classification. A survey of crawling techniques appears in [57].

3.2 Offline Data Processing and Content Management

Offline data processing for a search engine collects and parses web documents to produce a searchable index. The additional information collected in online database generation includes ranking features extracted from page links, anchor text, and user query log data. Content management organizes an online index with classification and partitioning, it also guides the crawling system to discover and refresh web documents. Some search engines have large databases with tens of billions of documents while others have comparatively smaller ones with a few hundred millions of pages, designed to handle a certain type of queries.

3.2.1 Document Parsing and Index Compression

Document parsing breaks apart the components of a document to form indexable tokens. Tokenization is straightforward for English documents while it is challenging for documents of other languages such as Chinese, Japanese or

Arabic. In those languages, words are not clearly delineated by white space. Natural language processing techniques can identify boundary of words with a good accuracy. A parser also detects the language of a document and its format because documents do not always clearly or accurately identify their language. Since documents do not always obey the desired format or syntax, a parser has to tolerate syntax errors as much as possible.

In tokenizing a document, there is an option to stem words and remove stop words. Stemming is the process for reducing inflected words to their base form. For example, a stemming algorithm [60, 45] reduces the words "fishing", "fished", "fish", and "fisher" to their root word "fish". Stemming can produce an improvement in ranking by matching more relevant documents while it can sometime cause a relevance issue. For example, people search for "fishing" will be less interested in types of "fish". Thus stemming has to be applied conservatively and one example is to restrict stemming and only derive plural forms for a selected set of keywords. Stemming is a language-specific technique and some languages such as Chinese have no or little word variations while stemming for languages such as Arabic and Hebrew is more complex and difficult. Stop words are those words which contribute little on the topic of a document. The examples of stop words in English include "the" and "who". Removing stop words explicitly from an index can impact search relevance because a stop word may carry a meaning in some cases. For example, "the who" is an English rock band. Typically a search engine indexes every word while incorporating removal of some stop words during query-time processing when appropriate.

Given a set of tokens (or called *terms*), a search engine normally represents a searchable database using a data structure called *the inverted index*, and a survey of inverted indexing can be found in [87]. An inverted index consists of a list of terms in the vocabulary of the data collection. For each term, the inverted index maintains a list of documents that contain such a term. This list of documents is often called *a posting* of the corresponding term. For example, given two documents $d_1 = \{a, b, c, a\}$ and $d_2 = \{c, e, c\}$, the inverted index is

$$a \to \{d_1\}, b \to \{d_1\}, c \to \{d_1, d_2\}, e \to \{d_2\}.$$

In addition to document identifications, a posting can store other information such as frequency of a term and positions of this term in each document. For example, the above inverted index can be augmented as

$$a \to \{d_1, 2, (1, 4)\}, b \to \{d_1, 1, (1)\}, c \to \{d_1, 1, (3), d_2, 2, (1, 3)\}, e \to \{d_2, 1, (1)\}.$$

The meaning of the posting for term "a" is interpreted as follows. This term appears in document d_1 with frequency 2 at positions 1 and 4.

The inverted index for a large web dataset takes a huge amount of space, and compression of the index data is necessary to enhance engine's online performance and reduce hosting cost [55, 2]. Typically document identifications in a posting are stored in an increasing order, and term positions of each document are also arranged in an increasing order. To compress a sequence of these numbers in a posting, a transformation called *delta encoding* is applied first to represent these numbers using their difference gap. For example, given a list $\{23, 40, 104, 108, 200\}$, this list can be reformulated as $\{23, 17, 64, 4, 92\}$. The first number of this list is not changed. The second number is the gap between the second number and the first number. The average number in the reformulated list is much smaller than that in the original list. This motivation for using delta encoding is that compressing small numbers is easier.

The next step of compression is to represent these gaps using an encoding method. A bit-aligned encoding method is called *Elias-*γ [26] which encodes each number G using a pair of two numbers (b, r). They satisfy $G = 2^b + r$ and $r = G \mod 2^b$. Next we discuss how to store the pair (b, r). The first number is stored as an unary code with b digits of 1s. The second number is stored as a standard binary number with b bits. Bit "0" is inserted to separate these two numbers. For instance, $17 = 2^4 + 1$; the γ code for 17 is 111100001. The leading 4 digits are all 1s, representing 2^4. Offset 0001 is found after the first 0 is encountered. In general, for each number G, this method costs $2 \log G + 1$ bits to encode. Thus γ encoding is only attractive to compress small numbers. To improve the effectiveness in compressing a larger number, the concept of δ encoding can be applied recursively to the first component b in the above expression. Namely $G = 2^{2^b + \delta} + r$. This method is called *Elias-*δ method which costs approximately $2 \log \log G + logG$ bits for each number G.

A bit oriented encoding method has its weakness in performance because computer operations are relatively more efficient towards byte-oriented programs. A simple byte-aligned method is called *v-byte* [75] that uses one byte to represent G if $G < 128$, and two bytes or more bytes for a bigger number. Seven bits of each byte store the content of G while 1 bit of such a byte indicates whether this byte is the last byte or not for the corresponding number. A comparison of compression algorithms can be found in [10, 80].

3.3 Content Management

Content management plays the following roles in offline data processing.

- Organize the vast amount of pages crawled to facilitate online search. For example, the online database is typically divided into content groups based on language and country. For each content group, there is a further division of content by tiers based on quality. Such a division also affects the crawling decision to prioritize on crawling targets and frequency.
- Perform in-depth analysis for better understanding of information represented in a document. Examples of such analysis are: recognizing document title, section titles, and highlighted words that capture the meaning of a document with different weights; identifying the structure of a page and recognizing site-menus which do not contain primary material; and interpreting Javascript embedded in a page to capture the true content of the page.
- Collect additional content and ranking signals. While on-page text features are important for a search engine, critical ranking information can also be gathered from other sources. For example, user search log provides a feedback on how users interact with search results given their queries. There is a variety of special information available on the Web and for instance, movie titles and show schedule. Such information can be organized for answering a class of queries or vertical search. For example, Ask.com has built a large database containing question-answer pairs extracted from the Web and used this index for matching and answering live question queries.

An online database is often divided into multiple tiers and multiple partitions. One reason is that the capacity constraint of a data center often restricts the size of a database index that can be hosted. Another reason is the requirement for geographical data distribution. Commonly-accessed content hosted close to users' location in a targeted market can reduce the engine's response time. In a multi-tier online database, the first tier usually contains frequently visited content of good quality. Even though there are tens of billions of pages available on the Web, a query from a vast majority of users can be answered by using a small percentage of these pages. The lower tiers of an online database contain pages which are less likely to be used to answer queries. To partition content, one can exploit characteristics of pages such as reputation and the historical visit frequency.

Anti-spamming is an important task of content management to detect pages with malicious attempts to influence the outcome of ranking algorithms. For example, some web sites present different content to a search engine's crawler compared to what is presented to users' web browsers. These web sites differentiate a crawler with a web browser based on the IP addresses or the User-Agent header of the HTTP request. This spamming technique, called *web cloaking*, is often used by low-quality sites to deceive search engines. Other spamming tactics include link farms, invisible text with stuffed keywords, and exploitation through user generated contention such as blogs. Automatic or semi-automatic techniques can be employed to identify spamming pages by examining content features, user browsing patterns, and hyperlink structure.

Search content management has adopted various classification techniques based on document features. The supervised learning techniques [54] fits well for this purpose using a set of labeled data to train a classifier that can label future cases. The examples of automatic or semi-automatic classification tasks include language categorization (e.g. English vs. German), country classification (e.g. United States, United Kingdom, and Canada), and spam detection. Another use of classification is to aid online result ranking for intent matching or result diversification. Page segmentation was conducted in [82] to detect semantic content structure in a web page. While text features represent the important characteristics of a web age, link structure among pages and feedback from a search log dataset also provide critical information in making classification decisions. Surveys of web page classification and text categorization appear in [65, 61]. A survey of related techniques can be found in [13].

3.3.1 Duplicate Elimination

A large number of pages crawled from the Web have near-identical content. For example, several studies [27, 7] indicate that there are over 30% of near duplicates in crawled pages. Search services need to return the most relevant

answers to a query and presenting identical or near-identical results leads to bad user experience. By removing duplicates, search result presentation would be more compact, and save users' time to browse results. Near duplicate detection can also assist low quality page removal. For example, by identifying a set of spam pages with certain patterns, other pages that are similar to these pages are the candidates for being classified as spam pages.

There are two types of duplicates among web pages. The first type of duplicates is caused by page redirection when one web page redirects to another, either through permanent or temporary HTTP redirect. Although the source and destination of redirection have identical contents, the crawler may visit them at different times and download different versions. Our experience is that up to 10% of duplicates in a large-scale database can be redirection-based. The second type is content-based when two web pages have near-identical information. This is measured by the percentage of content overlapping between the two pages. A popular way to model content similarity of two documents is to represent each document using shingles [8]. Let each document contain a sequence of terms, and each k-gram shingle contains k consecutive terms in this document. For example, let $D = \{a, b, c, e, f, g\}$, and its 2-gram shingle set is $D^2 = \{ab, bc, ce, ef, fg\}$. Let A^k and B^k denote a set of k-gram shingle signatures derived from pages A and B respectively. The Jaccard similarity of these two documents is

$$Sim(A, B) = \frac{|A^k \cap B^k|}{|A^k \cup B^k|}.$$

An English web page on average has a few hundreds of words, and computing the Jaccard similarity of web pages is expensive when computation is conducted for every pair. The minhash technique can be used to speedup with an approximation. The minhash value with respect to a random permutation function R is

$$MinHash(A^k) = \min_{x \in A^k}(R(x)).$$

It has been shown that $Sim(A, B)$ value is the same as the probability that $MinHash(A^k) = MinHash(B^k)$. Thus $Sim(A, B)$ can be approximated by using n random permutation functions and by computing the overlapping percentage of the corresponding n $MinHash$ values.

In general duplicate removal can be conducted during online query processing or in an offline processing stage. Removing redundant content in an offline system requires a very high precision in duplicate judgment. Fast approximation algorithms [8, 52, 32, 16, 19, 44, 73] are often not sufficient to simultaneously deliver high precision and high recall. A combination of these algorithms with additional features can improve precision (e.g. [35]). As a result, processing cost increases. Major search engine companies have often taken a compromised approach: while a relatively small percentage of duplicates can be removed conservatively in the offline process, a large portion of duplicates are kept in a live online database. During online query processing, duplicates are detected among top results matching a query, and only one result from duplicate pages is shown in the final search results. Such an approach simplifies offline processing, but increases the cost of online serving. To develop cost-conscious search applications, duplicates need to be removed in an offline stage as much as possible [86].

3.3.2 Distributed and Incremental Index Update

It is time consuming to process a large web data collection; this affects its freshness because data is continuously updated. Parallel processing using MapReduce [25] or other programming platforms can be used. Handling a large database from scratch still takes too much time, even in parallel on a large computer cluster. It is more effective to perform incremental update of the online database.

Figure 3 illustrates the architecture of a large-scale offline system we have developed in the past for incremental processing of billions of web pages. The key service modules of this system include page repository, update manager, URL property, online content controller, indexer, link manager, and content analysis subsystem. Each of them is running on a cluster of machines.

- Once a set of crawled pages is received, the update manager consults with a URL property service and runs a quick classifier to determine whether these pages are new or have significant content update. If appropriate, these new or updated pages are routed to the online database controller, link manager, and content analysis manager for further data update.

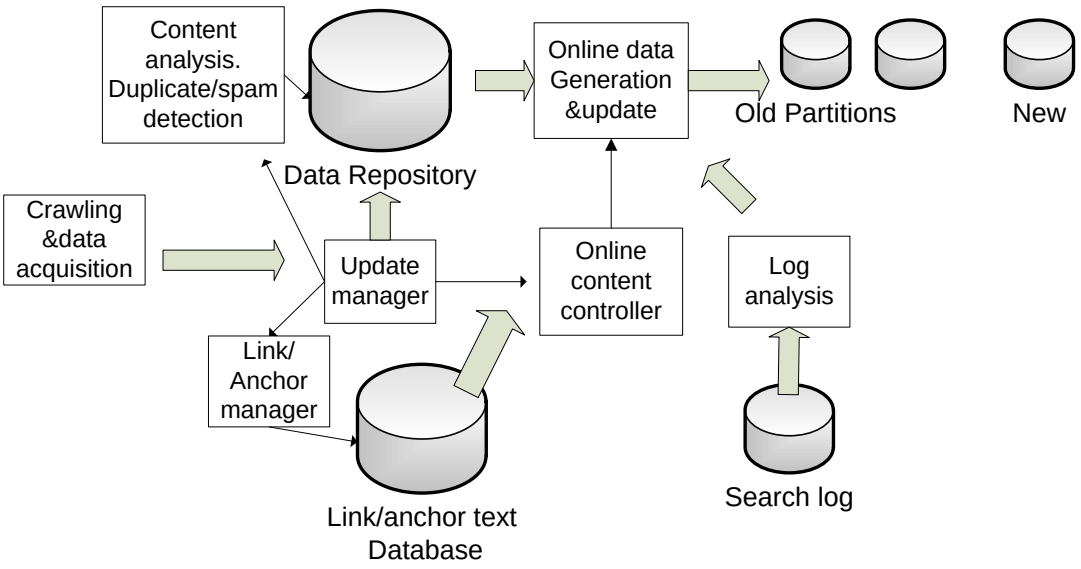

Figure 3: Architecture of an offline data processing system with incremental update.

- The online content controller manages the online partition structure. It decides when a new partition should be added to the online service and notifies that some URLs in the existing online partitions are stale.
- The indexer module fetches pages of the desired partition to construct an inverted index; it creates a partition to include new or updated URLs. Noted that the same URLs in the existing partitions will be marked as stale.
- The link manager maintains hyperlinks among web pages and summarizes anchor text from incoming links to a document. The content analysis subsystem detects duplicates incrementally, and it conducts classification and other mining tasks.

Various underlying system support is incorporated for implementing the above pipeline. That includes a middleware for managing continuously running services on a large cluster of machines, key-value stores with fast disk I/O [15] for page repository, high-throughput property services with a distributed hash table [85], and distributed message queuing for inter-component coordination.

The above approach for incremental index has an advantage that online data management is simplified: the modification to the existing partitions only needs to mark staleness when pages are updated. The online content controller makes a decision to remove an online partition when it contains too many stale URLs. Another approach for incremental index update is to merge the old index with new index [48]. The search system support for notification-based data updating is discussed in [59].

4 Query Processing and Online Architecture

An online system of a search engine typically contains the following components.

- 1) Query processing. This step classifies a query and rewrites it for better matching of relevant results.
- 2) Document retrieval. That is to search the inverted index distributed in a cluster of machines and to collect relevant results.
- 3) Result ranking. Given a large number of matched relevant results, this step filters out low quality results and ranks top results.
- 4) Snippet generation. This step provides a description for each URL to be shown in a search engine results page (SERP).

- 5) Result aggregation and displaying. Search results may be aggregated from multiple databases including the advertisement engine; this step merges these results and organizes them in a SERP.

4.1 Query Processing with Classification

Query processing classifies a query and rewrites it if needed with the goal of closely matching the searcher's intent. Examples of query classification include detecting query language and user location, and identify whether this query demands time-sensitive results or entity-oriented information such as a person, an organization, a movie, or a commercial product. The classification of a query helps the search engine to prioritize certain types of results in the final ranking, and it can also gives an indication if highly relevant results from some vertical channels should be included. For example, a query "white house" can trigger the inclusion of news, map, and image search results.

Common query rewriting operations involve stopword elimination, case folding, and spell correction. Stopword elimination detects the use of some popular terms which may not be able to effectively reflect the intention of a user. For example, terms "the" and "of" may be removed from query "the address of white house". Spell handling involves the examination of popular spell errors to suggest a correction. Case folding commonly used in web search engines does not differentiate cases. Case-insensitive search is more acceptable to users as it includes more relevant results to match users' queries without differentiating cases.

The query rewriting process may be explicit or implicit to users, which can be categorized into the following three situations.

- Suggestion without query rewriting. A search engine may not change query terms, but recommend a suggestion for correction. That is especially useful when the engine is not certain if the suggested correction is necessary for most of users.
- Explicit alteration. For certain queries, the search engine may automatically alter user queries while informing users the correction incorporated.
- Implicit alternation. For many queries, a search engine may automatically rewrite without informing users. It may incorporate multiple rewritings and aggregate results based on an internal ranking method. For example, the results for query "NJ tax return" may be supplemented with some additional results from "new jersey tax return". Rewriting has to be conservative based on query context. For instance, while the short form of state "Virginia" is VA, " VA" can also stand for "U.S. Department of Veterans Affairs".

4.2 Page Partitioning and Retrieval

The index data of a search engine is typically partitioned and distributed in many machines at one or possibly multiple data centers. There are two ways to distribute data to multiple machines: term-based or document-based.

- Document-based partitioning divides the index data among machines based on document identifications. Each machine contains the inverted index only related to documents hosted in this machine. During query processing, a search query is directed to all machines that host subsets of the data.
- Term-based partitioning divides the index based on terms and assigns postings of selected terms to each machine. This approach fits well for single-word queries, but it requires inter-machine result intersection for multi-word queries.

Document-based partitioning has several advantages. 1) It simplifies the database update process. When documents have a change of their content, only few machines that host these documents are affected. 2) It allows for good load balancing as the term distribution is more skewed. 3) It is more resilient to faults compared to term partitioning. If some machines fail, only the index of documents on these failed machines becomes unavailable, and the search engine may still be able to serve relevant results from other machines. For term-based partitioning , the failure of one machine could lead to more noticeable errors for certain queries which contain terms hosted in the failed machines.

The disadvantage of document-based distribution is that all machines need to participate in query answering. Thus the serving cost per query is more expensive. A combination of document and term based partitioning would be an option to exploit in hosting a large index database.

After the initial query processing, a search engine accesses the index following the semantics of a query. Be default, query terms are considered to be conjunctive. Namely, a document is considered to be a match if and only if it contains all query terms. In a disjunctive mode, a document is considered to be a match if only one query term presents in the document. While a search engine often provides an advanced search feature which support a combination of conjunctive and disjunctive search, the common usage of user queries follows a conjunctive format due to its simplicity.

Since the index is distributed to machines using document or term based partitioning, each of these machines returns a set of documents fully or partially matching the processed query; therefore additional processing is needed. For example, for the query "Britney spears", the posting of term "Britney" is intersected with the posting of term "spears". If two postings are hosted in different machines, inter-machine communication is necessary to accomplish this intersection. For document-based partitioning, the posting intersection is done within a machine, and each machine will report a subset of distinct documents that match this query. Once the search engine gathers matched documents, it ranks these matched documents and presents top results to users. We discuss page ranking in the next section.

Retrieval of relevant results during index matching is time consuming when searching a large dataset, and various techniques are developed to speedup this process. One technique is to divide the list of documents from the posting of a term into multiple chunks and add skip pointers in the data structure [56, 71]. In performing conjunctive search among postings for multiple query terms, the maximum document identification of a chunk can be used to determine if a chunk can be skipped when intersecting with a document identification from another posting. A skipping pointer for a chunk can be inserted into the data structure of inverted index so that the scanning of this chunk can be easily avoided.

4.3 Result Caching and Snippet Generation

Caching is a commonly adopted technique to speedup the query answering process because there are often over 50% of queries that can use previously-computed search results. Caching provides load smoothing to deal with traffic spikes caused by hot events during which different users often type the same or similar queries again and again to check the latest development. A search engine may adopt a variety of hierarchical caching strategies. For example, cache the top HTML response with snippets, cache the top results without snippets, cache a matched URL list in a local machine before result aggregation, and cache the postings of frequent query terms.

Given a limitation on cache size, results for most frequent queries can be cached, and some results may be pre-cached in anticipation of its popularity. Other replacement policies may consider weighted factors such as recomputing cost [30]. A key issue to consider for caching is freshness of cached results. As the search index is updated continuously, the concern can arise for the staleness of cached results. For example, a query that contains news results may not be cacheable because a later query needs to be answered with the latest news results. One strategy is to cache results only for relatively stable answers.

When top search results are presented to users, each document is described with a summary of its content, which is known as a snippet. There are two categories of techniques for document summarization. One is query-independent static snippet where a document is summarized without knowing queries. The second category is query-dependent dynamic summarization in which a snippet is composed of several short pieces of text, and each short piece often contains query terms. The selection of these text pieces depends on a number of features including the percentage of query terms covered, positions of text pieces in a document, importance of query terms covered, and the readability of the summary. A study on snippet readability was done in [42]. Producing high quality snippets is time-consuming, and efficient techniques for snippet generation were discussed in [74].

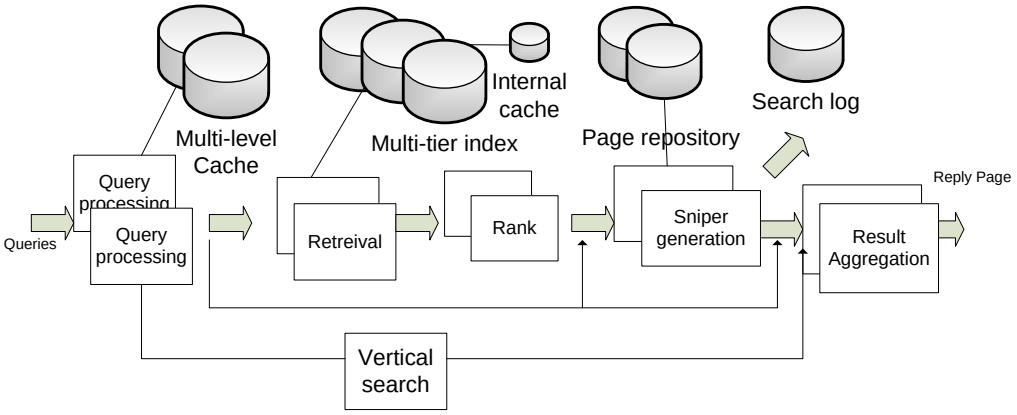

Figure 4: An online web search architecture.

4.4 An Example of Online Search Architecture

Figure 4 illustrates an online search architecture we have developed in the past in hosting billions of web documents and answering queries from tens of millions of queries every day. The database is divided following language and country classification of documents, and the query processing module can route a query to a proper collection of index based on the targeted user region. For example, English queries are processed using English data partitions. There is a small percentage of queries which contain mixed languages; for those queries, multiple language content groups are involved in search.

Once a query is received by the online system, this query is distributed to one of query processing front-ends through a load balancing traffic switch. The query processing module consults a multi-level cache to see if this query can be answered using the previously-computed result with or without snippets. If a query cannot be found from the cache, the page retrieval module searches URLs in Tier 1 of the online index and visits Tier 2 (or other lower priority tiers) only if the results matched from Tier 1 are not satisfactory. Our experience is that upto 70% of queries can be handled by Tier 1 without searching other tiers. There is an internal cache deployed for each partition so that retrieval results for frequent queries or compute-intensive queries are cached within each partition. There is also an aggregation module that combines results from multiple partitions and eliminates duplicates when needed, and this module can be deployed hierarchically also [20].

Ranking is conducted hierarchically. The page retrieval module selects top results from each partition, and these results are combined in an aggregation module which performs additional filtering. Signals for ranking are discussed in the next section. Once top ranked results are derived, the snippets for these results can be computed by accessing the text content of these documents from a page repository. There is a heavy I/O activity involved in accessing disk data for snippet generation, and caching snippets for popular queries can be used to improve the performance.

In addition to finding results from the HTML page collection, a query is also routed to vertical search channels including the advertisement engine and other types of popular web collections such as news and blogs, maps, images and videos. The top results from multiple channels are combined in the SERP. Optimizing the selection and placement of these results from different channels in a SERP is critical for users' experience with a search engine. Once a user browses URLs listed in the SERP, the search engine keeps a log on the click activity of this user. The user behavior extracted from this log provides a feedback for adjustment of ranking of results, triggering and placement of vertical search.

Each logical system component in Figure 4 runs on a large-scale computer cluster. As we mainly use a document-based distribution for search index, processing of each search request goes through thousands of machines. System support for the above engine includes request and internal traffic scheduling, fault tolerance, and replication support; the middleware and related system techniques were discussed in [67, 66, 20, 84].

5 Ranking

Ranking plays the most visible role in a search engine since most users only look at top results. Ranking can be conducted in a centralized manner or hierarchically so that each subsystem selects the best results first before submitting selected results for the next round of ranking. Relevance scores for documents are computed by exploiting their text and authority features, which are often called *ranking signals*.

We first give an overview of query-dependent and query-independent ranking signals, and describe feature analysis based on text, link, and click data. Then we will discuss score aggregation by using these signals.

5.1 Ranking signals

There are four aspects of ranking in matching the intent of a user.

- 1) Relevancy. Top documents need to be relevant to a user query.
- 2) Authoritativeness. High quality content is normally preferred since users rely on trustful information to learn or make a decision.
- 3) Freshness. Some queries are time sensitive and latest information is more appropriate for such type of queries.
- 4) Preference. Personal or geographical preference can also impact the choices of answers.

Rank signals are designed to reveal those aspects from the query or document point of view, and we summarize two categories of ranking signals as follows.

Query-dependent ranking signals.

- **Document text.** This feature measures the closeness between query words and what appears in a document. A web document can be divided into multiple text sections such as document title and body, and query words matched in different sections carry different weights. Additional weighting can be considered if query terms are highlighted in a document. Proximity, positions, and frequency of query words matched in a document are a prominent indicator of relevance.

  Some of text features in a web document are not visible to users; for example, description and keywords tags of an HTML document. Spammers often exploit such a feature to present irrelevant keywords; thus such features have to be used with caution.

- **Anchor text.** Anchor text of a web page appears in the hyperlinks of other web documents referring to this page. There can be many hyperlinks pointing to the same web document, and some of anchor text can be meaningless or of low quality. The commonality in anchor text and popularity of source web pages can be helpful in determining quality of anchor text.

- **URL text.** Text of a URL, especially when appearing in a position within or close to the host name, may sometime reveal the semantic of its content. For example, "download" in http://www.microsoft.com/en-us/download/ is an important keyword for matching query "Microsoft download".

- **Historical click-through data.** A search engine collects the behavior data from users when they search and browse matched results. These URLs clicked by a user following a search query are often relevant to such a query.

- **Query classification and preference.** Implicit features of a query can be extracted during classification to select preferred documents. For example, the language of a query prioritizes the language type of web documents to be matched. The originating geographical location of a query may direct the ranking system to select more content from this region. If a query matches a hot news topic, time-sensitive fresh content is preferred.

- **Link citation.** This feature uses the web hyperlink connectivity among those pages whose text matches a given query. A page cited by many other matched web pages can be a good indicator of authoritativeness for this query.

Query-independent signals.

- **Link popularity.** This feature analyzes how pages link to each other and then uses this information to determine the importance of each page on the web. If a page is cited from a large number of pages, it is often ranked more prominently.
- **Documentation classification and spam analysis.** A classification of web documents can provide a signal in matching user preference. For example, users are more likely to be looking for content in their own language than in other languages. File types such as HTML and Microsoft Word can also be a preference for users. Since spam analysis leads to a rating of page quality, a spam score is also a useful quality indicator.
- **URL Preference.** In many cases, users prefer a URL with a short length and prefer a static URL instead of a dynamically-generated URL. Dynamic generated URLs tend to change content from time to time, and their average quality is often lower than that of static URLs.
- **Site or page properties.** Some additional information on a web page or its site can be an indication of web page quality. For example, frequency of user visits, the time duration of visit, and serving speed or reliability of the web host.

There are many methods for modeling above ranking signals, and we next describe some of techniques for analyzing text, links, and click data.

5.2 Text analysis

The goal of text analysis is to quantify the relevance of a document based on how query words appear in this document. Boolean and phrase operators are available in a search engine for advanced retrieval semantics. For example, "Retrieval" OR "Search". Since most people use simple and short queries, we focus on text analysis of a search query with conjunctive semantics.

There are three issues in developing a formula measuring the text closeness of document-query matching.

- What are basic units of information to match a query with a document? A document is normally tokenized a sequence of words after excluding HTML tags, and each word is a basic term for matching. Thus a document can be considered as a bag of words. On the other hand, the order and closeness of terms plays an important role. For example, "Paris Hilton" is the actress name while "Hilton Paris" could mean the Hilton hotel at Paris.
- What is the proximity of search words appeared in a document? When search words appear close to each other in a document, this document is often considered to have a high relevancy. To capture word proximity, an n-gram term can be considered, which is defined as a consecutive sequence of n words in a document. An n-gram of size 1 is referred as a "unigram", size 2 is a "bigram", and size 3 is a "trigram".

  Another model for proximity is to use a text *span* [70, 72]. A document is segmented into multiple disjoint spans that cover search terms. Each span may not contain every query term and it is a minimum text window containing search terms as much as possible. Each span starts with the position of the first query term in a document and contains consecutive query terms in a document. This span ends when the distance between the current query term match position and the next query term match position exceeds a threshold. This span also ends if the current and next matches are the same term, or the previous query term is the same as the next query term.
- How are terms weighted? One method is to use a simple weighting based on where they appear. If a query term appears in the body of a document, it will receive credit. When it appears in the title, it receives weighted

credit. Another method that uses the frequency information is known as TF-IDF weighting (Term Frequency and Inverse Document Frequency). In this scheme, the importance of a term increases when it appears many times in a document. If a term appears many times in many documents, its weight decreases because it becomes less effective in differentiating a relevant document from a non-relevant document. A common formula for TF-IDF is defined as follows.

$$tfidf(t,d) = tf(t,d) * \log \frac{n}{df_t}$$

where n is the number of documents in the collection, $tf(t,d)$ is the occurrence count of term t in document d, and df_t is the number of documents that term t appears. Because denominator df_t may be zero, it is often adjusted as $1 + df_t$. Additionally, $tf(t,d)$ is sometime normalized against the length of document d. Variations of the TF-IDF weighting have been suggested in the literature, for example, Okapi BM25 [39].

- How can the closeness of matching be represented by aggregating weights from multiple query terms? One model is to use the cosine-based similarity function. Let document d contain a set of terms with weight w_i for each term, and let query q contain a set of query terms with weight qw_i. These weights are normalized, and the aggregated text score is

$$Score(d,q) = \Sigma_{t_i \in q} w_i * qw_i.$$

There are other ways to aggregate weights. Since text features can be represented in multiple ways to capture the appearance and proximity of search terms, they can be aggregated using a weighted summation function or a machine learning model. We discuss a weight-learning technique in Section 5.5.

5.3 Link Analysis

5.3.1 PageRank Algorithm

As a query-independent ranking signal, the earlier PageRank algorithm [58] measures the importance of a web document based on its presence in a large web graph. In this graph, each node is a web page and each edge is directed, representing a link from one page to another using the HTML hyperlink notation. The intuition underlying this link analysis is that highly linked pages are more important than pages with just few links, but not all links carry the same importance. A page with a link from a popular site may be ranked higher than other pages with more links but from obscure places.

Let n be the number of pages in a web graph, and let x be a page. A simplified version of the PageRank model can be expressed as:

$$R(x) = \frac{\lambda}{n} + (1-\lambda)\Sigma_{y \in P(x)} \frac{R(y)}{|S(y)|}$$

where $S(x)$ is the set of pages which x points to, and $P(x)$ is the set of pages that point to x.

PageRank computation can be conducted iteratively. Initially every page has its initial value as $\frac{1}{n}$. Then all pages will be assigned with a λ-linear combination of a rank value $\frac{1}{n}$ and additional rank credit propagated from its predecessors in the graph. After that, vector $R()$ is normalized. This process repeats until all nodes have a converged rank value.

Computing the in-degree of a web page can approximate its link popularity score, but this score can be less resilient to link spamming. That is because it is is relatively easy to create thousands of pages on the Web pointing to a page. The PageRank algorithm was initially viewed as the key to Google's differentiation compared to other search engines when Google was launched. Various modifications have been suggested in the literature to improve relevancy or speedup computation. For instance, see [40, 41].

5.3.2 HITS algorithm

The HITS algorithm [43] provides query-dependent scoring to determine the authoritativeness of a web page relevant to a query. The key difference compared to PageRank is that HITS computes link scores within a subgraph of a web graph derived based on a user query. The HITS algorithm derives two scores for every page. The first score is called *the authority score*, representing the authoritativeness of a page relevant to the subject of a user query. The second

score is called *the hub score* representing its rank value in serving as the compilation of relevant information. Let the authority score of a page x be $A(x)$, and let the hub score of this page be $H(x)$.

$$A(x) = \Sigma_{y \in P(x)} H(y) \text{ and } H(x) = \Sigma_{y \in S(x)} A(y)$$

where $P(x)$ contains the predecessors of page x in this graph, and $S(x)$ contains the successors of x.

The computation of these scores is performed iteratively. After an initial score assignment, all pages update their hub and authorities based on the above formula. These values are normalized at each iteration, and this process repeats until all pages have a converged value. Mathematically speaking, the above process computes the principal Eigen vector of matrix C^tC and CC^t where C is the connectivity matrix based on the link structure of the query graph, and C^t is the transpose of C.

The major challenge in using HITS is its online computation cost. There are several extension or followup work to improve the seminal work of PageRank and HITS. That includes topic distillation [5] and multi-community hub-authority [24, 31]. A fast algorithm for computing hub authority scores with sparse matrix approximation was developed at Teoma/Ask [81].

5.4 Click analysis

Users interact with a search engine by clicking search results of a query, browsing them, and reformulating their queries, and performing additional browsing. Such interaction is logged in a search engine, and the click data can be thought of as a tuple (t, u, q, c) indicating that user u clicks result c for query q at time t. While click data can be noisy, URLs clicked by users are likely to convey some information for relevance judgment. A document may be ranked high if it has attracted the greatest number of previous users for the same query. A log dataset can be divided into a sequence of search sessions for each user. Each user session contains a sequence of queries and results picked by this user. The following correlations among queries and URL results can be extracted from session analysis.

- **Query-to-pick (Q2P).** A Q2P correlation associates a query with a pick. When a search engine returns a result in response to a query, and a user picks this result, that constitutes a correlation candidate. When multiple independent users make the same association, that makes this Q2P correlation more plausible.

  As an extended association, after a user enters a query, picks recorded for a different query within the same user session can be associated with the initial query. Another extension is that different pages browsed as a result of a URL pick may also be associated with the original query. The above two extensions represent the fact that a user often reformulates a query or browses beyond a search link to look for right content. Such an approach carries data errors and additional noise filtering techniques need to be applied. For instance, a restriction can be applied using query similarity from one pick to another in a session, the order and depth of query rewriting or browsing steps, dwelling time, and the first and last choice of browsing.

- **Query-to-query (Q2Q)**. This correlation associates queries issued during a user session. The confidence for such association depends on various factors including query similarity, the issuing time between queries, the number of intervening queries or picks, the order of association, and the number of sessions sharing such an association. The Q2Q correlation can help the generation of query suggestions in search engine response pages. For example, "electronic eavesdropping" is related to a query "eavesdropping devices". Misspelled "Hotel Mariot" is related to "Hotel Marriott".

- **Pick-to-pick (P2P)**. This correlation associates picks issued during a user session, which is analogous to the Q2Q correlation described above. The P2P correlation can reveal a set of URLs that are similar or related for certain topics. As an example, toyota.com is the top answer for answering " toyota car", honda.com and ford.com may also be listed as similar results in case users are interested in exploiting similar products.

While click-though data can be very noisy and incomplete with a sparse structure, it still provides valuable information for search quality improvement. Ask.com which bought DirectHit.com in 1999 was the earlier system that ranked search results based on URLs' clickthrough rates and the adjustment is applied when URLs with a relatively lower rank may receive more clicks compared to those with a higher rank. Ask.com further adopted session-based click data

to improve ranking with Q2P and Q2Q correlations [23]. The use of click data to derive user's relative preference was studied in [38] to train a ranking function. Query reformulation that expands the Q2P correlations in learning to rank was considered in [62]. The relevancy impact of integrating a number of click-based features was demonstrated in [1]. The bipartite graph structure of query-document relationship from a query log for modeling Q2P, Q2Q, and P2P correlations was studied in [79, 21]. Integrating browsing activities of a search session for ranking was considered in [68].

5.5 Score Aggregation and Learning to Rank

There are many ranking signals that can be extracted or characterized for each matched document in responding to a query. Those signals need to be aggregated, and their weights can be imposed empirically or derived using a machine learning method. Aggregation of these signals can be performed hierarchically. In this way, ranking signals can be divided into several semantically-relevant components for fine tuning.

Machine-learned ranking uses supervised or semi-supervised techniques that automatically construct a ranking model from training data. Training data consists of a set of queries, and each query has a list of documents with a relevance label. The example of a relevance label is "relevant", "related", or "irrelevant" for a three-level judgment. There are various ways to model a learning function based on a training dataset, and multiple models can be combined by an ensemble method such as bagging and boosting [6, 29, 28] to further improve the learning accuracy.

We illustrate a simple learning function as follows using the RankSVM method [36, 38]. Let $\vec{x_i}$ be a feature vector representing a set of ranking signals for document d_i. An aggregated score function $f(x_i)$ is a product of a weight vector $\vec{w}$ with $\vec{x_i}$. Function $f()$ defines ranking as follows: if $f(x_i) > f(x_j)$, document d_i should be ranked before d_j in answering a query. This condition implies that we need to find $\vec{w}$ so that $\vec{w} \times (\vec{x_i} - \vec{x_j}) > 0$.

Following the above condition, a training dataset is converted based on the pairwise relationship of documents. Assume that page d_i with a feature vector x_i and another page d_j with a feature vector x_j are labeled for answering the same query. Define a new label $e_{i,j}$ for pair (d_i, d_j) as 1 if d_i should be ranked before d_j and -1 if d_i should be ranked after d_j. The new training data can be thought of as a set of triplets. Each triplet is $(Q, \vec{x_i} - \vec{x_j}, e_{i,j})$, where Q is a test query and d_i and d_j are from the labeled result list of Q. Then the hard-margin linear SVM model is to minimize $||\vec{w}||^2$ subject to a constraint for all points in the training data: $\vec{w} \times (\vec{x_i} - \vec{x_j}) \times e_{i,j} \geq 0$.

The above process demonstrates that a ranking problem can be transformed to a classification problem, and other classification-based algorithms can also be used. Learning-to-rank algorithms have been categorized as point-wise, pair-wise, and list-wise [49]. The pointwise approach predicts the relevancy score of each document for a query using a supervised learning method such as regression and classification. The pairwise approach considers learning as a binary classification problem to decide if a document should be ranked prior to another document. Examples of this approach are RankSVM [36, 38], RankBoost [28], and RankNet [9]. The list-wise approach tries to directly optimize the cost function, following a relevancy evaluation measure. Examples of such an approach include LambdaMART [76], SVM^{map} [83], and AdaRank [78]. Machine-learned ranking has its advantage for leveraging a large training dataset while a hand-built model has its limitation, however, may sometime outperform an automatically-computed model through fine-grain tuning.

6 Search Engine Evaluation

6.1 Strategies in Quality Evaluation

A common evaluation for search engine quality is the relevance of search results as well as the speed in answering a query. It is not easy to define the metric for measuring the relevance of search results. This is because in many cases, it is hard to tell what a user really wants by viewing a query. Precise detection of user's query intent is the holy grail of search. For navigational queries such as Microsoft, most users want to navigate to the respective sites. Queries such "US open" can have multiple answers from time to time. Intent of some queries can be heavily biased by locations or languages used. For instance, CMU can mean Central Michigan University, Carnegie Mellon University,

or Central Methodist University, depending users' location.

One way to judge result relevance of a search engine is to employ a team of evaluators, and let them type queries collected from an engine log, and assess if the returned answer matches the typed query. This has a challenge since sometime an evaluator may not be able figure out the right answers. Another complexity is that today's search-engine users expect more than just result relevance, and there are multiple aspects of a search engine affecting the satisfactory degree of an end user. For example, are the results from authoritative sources? Are they free of spam? Are their titles and snippets descriptive enough? Are they fresh and timely? Does the answer include additional visual elements such as maps or images which may be helpful? A through evaluation needs to cover each of these dimensions when appropriate.

The search log data can be used to assess the quality of an engine and to capture the satisfactory degree of end users. When users engage a search engine and select presented results more frequently, this is a good indicator that users like this engine. The key metrics that can be derived from a search log include the daily or weekly user traffic, the visit frequency of users, result pick rate, and abandon rate when users do not select any result. The metrics derived from a search log carry certain noise. For example, result pick rate is often affected by the placement of results and interface design in a SERP. Site traffic can be also affected by seasonality. The interpretation of these metrics requires a deep data analysis from multiple aspects.

6.2 Evaluation Metrics

To measure whether an information retrieval system returns relevant results or not, precision and recall are the two metrics developed in the earlier studies. Precision is the percentage of returned results which are relevant, and recall is the percentage of all relevant results covered in the returned list. For web search, when there are many results relevant to a query, a user normally only browses top few results; thus precision is more critical than recall in such a setting. When there are only very few results relevant to a query, a recall ratio measures the ranking and crawling capability in finding these documents.

Another measure called *the Normalized Discounted Cumulative Gain (NDCG)* [37] which considers the graded relevance judgment and the usefulness of a document based on its position in the ranked result list. In this scheme, the result quality is judged using graded relevance with a multi-level label. For example, a six-level label can be "perfect", "excellent", "good", "fair", "poor", and "bad". The NDCG value starts with a following concept called *the Discounted Cumulative Gain (DCG)* at a particular rank position k:

$$DCG[k] = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{\log_2(1+i)}$$

where rel_i is the graded relevance value at position i. For example, 0 stands for "bad", 1 stands for "poor", 2 stands for "fair", and so on. The NDCG value normalizes the DCG value as

$$NDCG[k] = \frac{DCG[k]}{DCG'[k]}$$

where $DCG'[k]$ is the ideal DCG at position k. The $DCG'[k]$ value is computed by producing the maximum possible DCG till position k. The NDCG value for all queries is averaged to assess the average ranking quality.

The other evaluation metrics for a search engine includes freshness and coverage. To assess the freshness, one metric can be the time from creation of a document on the Web to the time when it appears in the online index. To measure the completeness, the size of database can be used. Since there are a large amount of low quality URls on the Web, the size alone is not sufficient. An additional strategy is topic-based sampling to measure the satisfactory degree in answering users' queries for selected topics.

The speed of the engine performance can be measured by the round-trip response time of a query. Typically a search engine is required to return a query result within a second, but in practice a search engine is able to deliver a search result within a few hundred milliseconds. During search engine capacity planning, the throughput of an online system is also measured so that a system can be optimized to answer as many queries as possible at every second. When the engine configuration such as the database size changes, its performance metrics may be affected and need to be examined.

7 Concluding Remarks

The main challenge of search engine technology is the handling of big data with significant noise. Search engine technology has been improved steadily in the last decade while the expectation of Internet users has also increased considerably. There are still a large number of unsolved technical problems to accurately capture the intent of search users. As more and more ranking signals are introduced to improve result relevance for different categories of queries, the interaction of these signals can sometime yield to subtle side effects. Thus, further improvement of search quality towards the next level requires more effort. In the next ten years, mobile and multimedia search is a key battleground among search providers. With popularity of social networking, social search is also another hot topic to facilitate information sharing. Search-based information technology will continue to advance and evolve in multiple directions, so Internet and mobile users can find the information that they need quickly.

Acknowledgments

We thank Teofilo Gonzalez for his valuable comments. This work was supported in part by NSF IIS-1118106. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Eugene Agichtein, Eric Brill, Susan Dumais, and Robert Ragno. Learning user interaction models for predicting web search result preferences. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '06, pages 3–10, New York, NY, USA, 2006. ACM.

[2] Vo Ngoc Anh and Alistair Moffat. Inverted index compression using word-aligned binary codes. *Information Retrieval*, 8(1):151–166, January 2005.

[3] Ricardo Baeza-Yates, Carlos Castillo, Mauricio Marin, and Andrea Rodriguez. Crawling a country: better strategies than breadth-first for web page ordering. In *Special interest tracks and posters of the 14th international conference on World Wide Web*, WWW '05, pages 864–872, New York, NY, USA, 2005. ACM.

[4] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval (2nd Edition)*. Addison Wesley, 2011.

[5] Krishna Bharat and Monika R. Henzinger. Improved algorithms for topic distillation in a hyperlinked environment. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '98, pages 104–111, 1998.

[6] Leo Breiman and E. Schapire. Random forests. In *Machine Learning*, pages 5–32, 2001.

[7] Andrei Z. Broder. Identifying and filtering near-duplicate documents. In *Proc. of Combinatorial Pattern Matching, 11th Annual Symposium*, pages 1–10, 2000.

[8] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. In *Proc. of WWW 1997*, pages 1157–1166.

[9] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 89–96, New York, NY, USA, 2005. ACM.

[10] Stefan Büttcher and Charles L. A. Clarke. Index compression is good, especially for random access. In *CIKM*, pages 761–770, 2007.

[11] Stefan Büttcher, Charles L. A. Clarke, and Gordon V. Cormack. *Information Retrieval: Implementing and Elvaluating Search Engines*. The MIT Press, 2010.

[12] Cassandra. http://cassandra.apache.org/.

[13] Carlos Castillo and Brian D. Davison. Adversarial web search. *Found. Trends Inf. Retr.*, 4(5):377–486, 2011.

[14] Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused crawling: a new approach to topic-specific web resource discovery. *Computer Networks*, 31:1623–1640, 1999.

[15] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: a distributed storage system for structured data. In *Proceedings of the 7th symposium on Operating systems design and implementation*, OSDI '06, pages 205–218. USENIX Association, 2006.

[16] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388, 2002.

[17] Junghoo Cho and Hector Garcia-Molina. Parallel crawlers. In *Proceedings of the 11th international conference on World Wide Web*, WWW '02, pages 124–135, New York, NY, USA, 2002. ACM.

[18] Junghoo Cho and Hector Garcia-Molina. Effective page refresh policies for web crawlers. *ACM Trans. Database Syst.*, 28(4):390–426, December 2003.

[19] Abdur Chowdhury, Ophir Frieder, David A. Grossman, and M. Catherine McCabe. Collection statistics for fast duplicate document detection. *ACM Trans. Inf. Syst.*, 20(2):171–191, 2002.

[20] Lingkun Chu, Hong Tang, Tao Yang, and Kai Shen. Optimizing data aggregation for cluster-based internet services. In *PPOPP*, pages 119–130, 2003.

[21] Nick Craswell and Martin Szummer. Random walks on the click graph. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 239–246, New York, NY, USA, 2007. ACM.

[22] Bruce Croft, Donald Metzler, and Trevor Strohman. *Search Engines: Information Retrieval in Practice.* Addison Wesley, 2010.

[23] Andy Curtis, Alan Levin, and Apostolos Gerasoulis. Methods and systems for providing a response to a query. *US Patent 7152061*, 2006.

[24] Brian D. Davison, Apostolos Gerasoulis, Konstantinos Kleisouris, Yingfang Lu, Hyun ju Seo, Wei Wang, and Baohua Wu. Discoweb: Applying link analysis to web search. In *Proceedings of the Eighth International World Wide Web Conference*, pages 148–149, 1999.

[25] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *Proc. of OSDI 2004*.

[26] P. Elias. Universal codeword sets and representations of the integers. *Information Theory, IEEE Transactions on*, 21(2):194–203, Mar 1975.

[27] Dennis Fetterly, Mark Manasse, Marc Najork, and Janet Wiener. A large-scale study of the evolution of web pages. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 669–678, 2003.

[28] Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 4:933–969, December 2003.

[29] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

[30] Qingqing Gan and Torsten Suel. Improved techniques for result caching in web search engines. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 431–440, 2009.

[31] Apostolos Gerasoulis, Wei Wang, and Hyun-Ju Seo. Retrieval and display of data objects using a cross-group ranking metric. *US Patent 7024404*, 2006.

[32] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.

[33] Hadoop. http://hadoop.apache.org/.

[34] HBase. http://hbase.apache.org/.

[35] Monika Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 284–291, 2006.

[36] Ralf Herbrich, Thore Graepel, and Klaus Obermayer. Large margin rank boundaries for ordinal regression. *Advances in Large Margin Classifiers*, pages 115–132, January 2000.

[37] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, October 2002.

[38] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 133–142, New York, NY, USA, 2002. ACM.

[39] K. Sparck Jones, S. Walker, and S. E. Robertson. A probabilistic model of information retrieval: development and comparative experiments. *Inf. Process. Manage.*, 36(6):779–808, November 2000.

[40] Sepandar Kamvar, Taher Haveliwala, Chris Manning, and Gene Golub. Extrapolation methods for accelerating pagerank computations. In *Twelfth International World Wide Web Conference (WWW 2003)*, 2003.

[41] Sepandar D. Kamvar, Taher H. Haveliwala, Christopher D. Manning, and Gene H. Golub. Exploiting the block structure of the web for computing pagerank. In *Stanford University Technical Report*, 2003.

[42] Tapas Kanungo and David Orr. Predicting the readability of short web summaries. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, WSDM '09, pages 202–211, 2009.

[43] Joh M. Kleinberg. Authoritative sources in a hyperlinked environment. In *Journal of the ACM*, pages 604–632, 1999.

[44] Aleksander Kolcz, Abdur Chowdhury, and Joshua Alspector. Improved robustness of signature-based near-replica detection via lexicon randomization. In *Proceedings of KDD*, pages 605–610, 2004.

[45] Robert Krovetz. Viewing morphology as an inference process. In *Proceedings of the 16th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pages 191–202, 1993.

[46] Hsin-Tsang Lee, Derek Leonard, Xiaoming Wang, and Dmitri Loguinov. Irlbot: scaling to 6 billion pages and beyond. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 427–436, New York, NY, USA, 2008. ACM.

[47] Lemur. http://www.lemurproject.org/.

[48] Nicholas Lester, Alistair Moffat, and Justin Zobel. Efficient online index construction for text databases. *ACM Trans. Database Syst.*, 33(3):19:1–19:33, September 2008.

[49] Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[50] Appache Lucene. http://lucene.apache.org.

[51] Jayant Madhavan, David Ko, Lucja Kot, Vignesh Ganapathy, Alex Rasmussen, and Alon Y. Halevy. Google's deep web crawl. *PVLDB*, 1(2):1241–1252, 2008.

[52] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. Detecting near-duplicates for web crawling. In *Proc. of WWW '07*, pages 141–150, 2007.

[53] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge Unviersity Press, 2008.

[54] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

[55] Alistair Moffat and Lang Stuiver. Binary interpolative coding for effective index compression. *Information Retrieval*, 3(1):25–47, July 2000.

[56] Alistair Moffat and Justin Zobel. Self-indexing inverted files for fast text retrieval. *ACM Trans. Inf. Syst.*, 14(4):349–379, October 1996.

[57] Christopher Olston and Marc Najork. Web crawling. *Found. Trends Inf. Retr.*, 4(3):175–246, March 2010.

[58] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. *Technical report, Stanford Digital Library project*, 1998.

[59] Daniel Peng and Frank Dabek. Large-scale incremental processing using distributed transactions and notifications. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–15, 2010.

[60] M. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[61] Xiaoguang Qi and Brian D. Davison. Web page classification: Features and algorithms. *ACM Comput. Surv.*, 41(2):12:1–12:31, February 2009.

[62] Filip Radlinski and Thorsten Joachims. Query chains: learning to rank from implicit feedback. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, KDD '05, pages 239–248, New York, NY, USA, 2005. ACM.

[63] robots.txt. http://www.robotstxt.org/.

[64] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[65] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, March 2002.

[66] Kai Shen, Hong Tang, Tao Yang, and Lingkun Chu. Integrated resource management for cluster-based internet services. In *Fifth USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[67] Kai Shen, Tao Yang, Lingkun Chu, JoAnne Holliday, Douglas A. Kuschner, and Huican Zhu. Neptune: Scalable replication management and programming support for cluster-based network services. In *3rd USENIX Symposium on Internet Technologies and Systems*, pages 197–208, 2001.

[68] Adish Singla, Ryen White, and Jeff Huang. Studying trailfinding algorithms for enhanced web search. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, pages 443–450, New York, NY, USA, 2010. ACM.

[69] Sitemaps. http://www.sitemaps.org.

[70] Ruihua Song, Michael J. Taylor, Ji-Rong Wen, Hsiao-Wuen Hon, and Yong Yu. Viewing term proximity from a different perspective. In *Proceedings of the IR research, 30th European conference on Advances in information retrieval*, ECIR'08, pages 346–357, Berlin, Heidelberg, 2008. Springer-Verlag.

[71] Trevor Strohman and W. Bruce Croft. Efficient document retrieval in main memory. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 175–182, 2007.

[72] Krysta M. Svore, Pallika H. Kanani, and Nazan Khan. How good is a span of terms?: exploiting proximity to improve web retrieval. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, pages 154–161, New York, NY, USA, 2010. ACM.

[73] Martin Theobald, Jonathan Siddharth, and Andreas Paepcke. Spotsigs: robust and efficient near duplicate detection in large web collections. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 563–570, 2008.

[74] Andrew Turpin, Yohannes Tsegay, David Hawking, and Hugh E. Williams. Fast generation of result snippets in web search. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 127–134, 2007.

[75] Hugh E. Williams and Justin Zobel. Compressing integers for fast file access. *The Computer Journal*, 42:193–201, 1999.

[76] Qiang Wu, Christopher J. Burges, Krysta M. Svore, and Jianfeng Gao. Adapting boosting for information retrieval measures. *Inf. Retr.*, 13(3):254–270, June 2010.

[77] Xapian. http://xapian.org/.

[78] Jun Xu and Hang Li. Adarank: a boosting algorithm for information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 391–398, 2007.

[79] Gui-Rong Xue, Hua-Jun Zeng, Zheng Chen, Yong Yu, Wei-Ying Ma, WenSi Xi, and WeiGuo Fan. Optimizing web search using web click-through data. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, CIKM '04, pages 118–126, New York, NY, USA, 2004. ACM.

[80] Hao Yan, Shuai Ding, and Torsten Suel. Inverted index compression and query processing with optimized document ordering. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 401–410, 2009.

[81] Tao Yang, Wei Wang, and Apostolos Gerasoulis. Relevancy-based database retrieval and display techniques. *US Patent 7028026*, 2006.

[82] Shipeng Yu, Deng Cai, Ji-Rong Wen, and Wei-Ying Ma. Improving pseudo-relevance feedback in web information retrieval using web page segmentation. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 11–18, 2003.

[83] Yisong Yue, Thomas Finley, Filip Radlinski, and Thorsten Joachims. A support vector method for optimizing average precision. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 271–278, 2007.

[84] Jingyu Zhou, Lingkun Chu, and Tao Yang. An efficient topology-adaptive membership protocol for large-scale cluster-based services. In *19th International Parallel and Distributed Processing Symposium (IPDPS 2005)*. IEEE Computer Society, 2005.

[85] Jingyu Zhou, Caijie Zhang, Hong Tang, Jiesheng Wu, and Tao Yang. Programming support and adaptive checkpointing for high-throughput data services with log-based recovery. In *Proceedings of the 2010 IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2010, Chicag*, pages 91–100, 2010.

[86] Shanzhong Zhu, Alexandra Potapova, Maha Alabduljali, Xin Liu, and Tao Yang. Clustering and load balancing optimization for redundant content removal. In *Proceedings of 22nd International World Wide Web Conference, Industry Track*, 2012.

[87] Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2), July 2006.