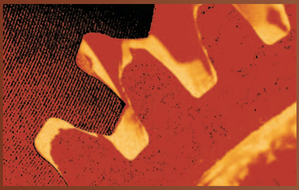
Web Search Engines: Part 1

David Hawking
CSIRO ICT Centre



A behind-the-scenes look explores the data processing "miracle" that characterizes Web crawling and searching.

In 1995, when the number of "usefully searchable" Web pages was a few tens of millions, it was widely believed that "indexing the whole of the Web" was already impractical or would soon become so due to its exponential growth. A little more than a decade later, the GYM search engines—Google, Yahoo!, and Microsoft—are indexing almost a thousand times as much data and between them providing reliable subsecond responses to around a billion queries a day in a plethora of languages.

If this were not enough, the major engines now provide much higher-quality answers. For most searchers, these engines do a better job of ranking and presenting results, respond more quickly to changes in interesting content, and more effectively eliminate dead links, duplicate pages, and off-topic spam.

In this two-part series, we go behind the scenes and explain how this data processing "miracle" is possible. We focus on whole-of-Web search but note that enterprise search tools and portal search interfaces use many of the same data structures and algorithms.

Search engines cannot and should not index every page on the Web. After all, thanks to dynamic Web page generators such as automatic calendars, the number of pages is infinite.

To provide a useful and cost-effective service, search engines must reject as much low-value automated content as possible. In addition, they can ignore huge volumes of Web-accessible data, such as ocean temperatures and astrophysical observations, without harm to search effectiveness. Finally, Web search engines have no access to restricted content, such as pages on corporate intranets.

What follows is not an inside view of any particular commercial engine—whose precise details are jealously guarded secrets—but a characterization of the problems that whole-of-Web search services face and an explanation of the techniques available to solve these problems.

INFRASTRUCTURE

Figure 1 shows a generic search engine architecture. For redundancy and fault tolerance, large search engines operate multiple, geographically distributed data centers. Within a data center, services are built up from clusters of commodity PCs. The type of PC in these clusters depends upon price, CPU speed, memory and disk size, heat output, reliability, and physical size (labs.google.com/papers/googlecluster-ieee.pdf). The total number of servers for the largest engines is now reported to be in the hundreds of thousands.

Within a data center, clusters or individual servers can be dedicated to specialized functions, such as crawling, indexing, query processing, snippet generation, link-graph computations, result caching, and insertion of advertising content. Table 1 provides a glossary defining Web search engine terms.

Large-scale replication is required to handle the necessary throughput. For example, if a particular set of hardware can answer a query every 500 milliseconds, then the search engine company must replicate that hardware a thousandfold to achieve throughput of 2,000 queries per second. Distributing the load among replicated clusters requires high-throughput, high-reliability network front ends.

Currently, the amount of Web data that search engines crawl and index is on the order of 400 terabytes, placing heavy loads on server and network infrastructure. Allowing for overheads, a full crawl would saturate a 10-Gbps network link for more than 10 days. Index structures for this volume of data could reach 100 terabytes, leading to major challenges in maintaining index consistency across data centers. Copying a full set of indexes from one data center to another over a second 10-gigabit link takes more than a day.

CRAWLING ALGORITHMS

The simplest crawling algorithm uses a queue of URLs yet to be visited and a fast mechanism for determining if it has already seen a URL. This requires huge data structures—a simple list of 20 billion URLs contains more than a terabyte of data.

The crawler initializes the queue with one or more "seed" URLs. A good seed URL will link to many high-quality Web sites—for example, www.dmoz.org or wikipedia.org.

Crawling proceeds by making an HTTP request to fetch the page at the first URL in the queue. When the crawler fetches the page, it scans the contents for links to other URLs and adds each previously unseen URL to the queue. Finally, the crawler saves the page content for indexing. Crawling continues until the queue is empty.

Real crawlers

In practice, this simple crawling algorithm must be extended to address the following issues.

Speed. If each HTTP request takes one second to complete—some will take much longer or fail to respond at all—the simple crawler can fetch no more than 86,400 pages per day. At this rate, it would take 634 years to crawl 20 billion pages. In practice, crawling is carried out using hundreds of distributed crawling machines.

A hashing function determines which crawling machine is responsible for a particular URL. If a crawling machine encounters a URL for which it is not responsible, it passes it on to the machine that is responsible for it.

Even hundredfold parallelism is not sufficient to achieve the necessary crawling rate. Each crawling machine therefore exploits a high degree of internal parallelism, with hundreds or thousands of threads issuing requests and waiting for responses.

Politeness. Unless care is taken, crawler parallelism introduces the risk that a single Web server will be bombarded with requests to such an extent that it becomes overloaded. Crawler algorithms are designed to ensure that only one request to a server is made at a time and that a politeness delay is inserted between requests. It is also necessary to take into account bottlenecks in the Internet; for example, search engine crawlers have sufficient bandwidth to completely saturate network links to entire countries.

Excluded content. Before fetching a page from a site, a crawler must fetch that site's robots.txt file to determine whether the webmaster has specified that some or all of the site should not be crawled.

Duplicate content. Identical content is frequently published at multiple URLs. Simple checksum comparisons can detect exact duplicates, but when the page includes its own URL, a visitor counter, or a date, more sophisticated fingerprinting methods are needed.

Crawlers can save considerable resources by recognizing and eliminating duplication as early as possible because unrecognized duplicates can contain relative links to whole families of other duplicate content.

Search engines avoid some systematic causes of duplication by transforming URLs to remove superfluous parameters such as session IDs and by casefolding URLs from case-insensitive servers.

Continuous crawling. Carrying out full crawls at fixed intervals would imply slow response to important changes in the Web. It would also mean that the crawler would continuously refetch low-value and static pages, thereby incurring substantial costs without significant benefit. For example, a corporate site's 2002 media releases section rarely, if ever, requires recrawling.

Interestingly, submitting the query "current time New York" to the

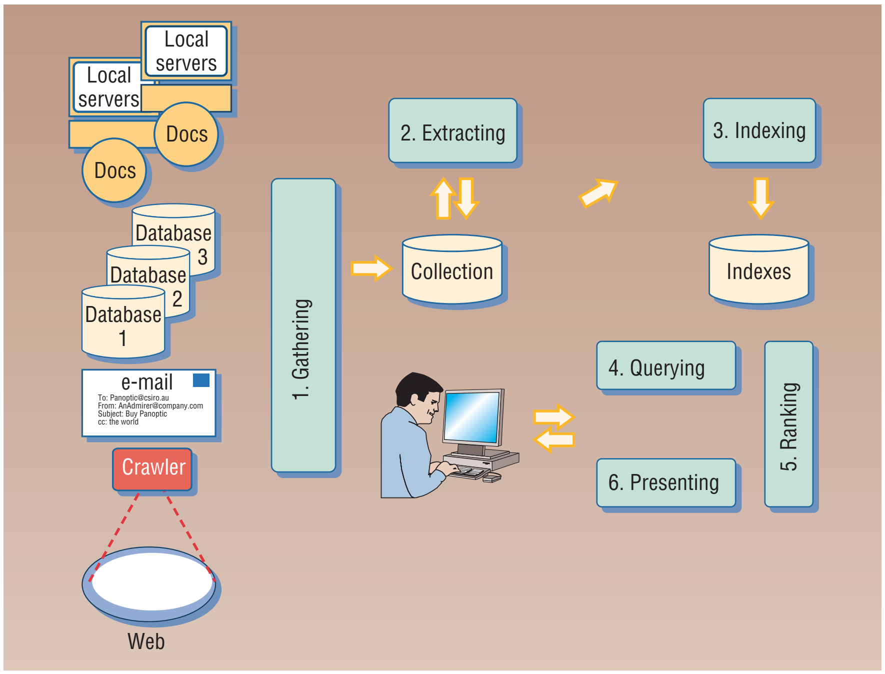


Figure 1. Generic search engine architecture. Enterprise search engines must provide adapters (top left) for all kinds of Web and non-Web data, but these are not required in a purely Web search.

Table 1. Web search engine glossary.

Term	Definition
URL	A Web page address—for example, http://www.computer.org.
Crawling	Traversing the Web by recursively following links from a seed.
Indexes	Data structures permitting rapid identification of which crawled pages contain particular words or phrases.
Spamming	Publication of artificial Web material designed to manipulate search engine rankings for financial gain.
Hashing function	A function for computing an integer within a desired range from a string of characters, such that the integers generated from large sets of strings—for example, URLs—are evenly distributed.

GYM engines reveals that each of these engines crawls the www.timeanddate.com/worldclock site every couple of days. However, no matter how often the engines crawl this site, the search result will always show the wrong time.

To increase crawling bang-per-buck, a priority queue replaces the simple queue. The URLs at the head of this queue have been assessed as having the highest priority for crawling, based on factors such as change frequency, incoming link count, click frequency, and so on. Once a URL is crawled, it is reinserted at a position in the queue determined by its reassessed priority. In this model, crawling need never stop.

Spam rejection. Primitive spamming techniques, such as inserting misleading keywords into pages that are invisible to the viewer—for example, white text on a white background, zero-point fonts, or meta tags—are easily detected. In any case, they are ineffective now that rankings depend heavily upon link information (www-db.stanford.edu/pub/papers/google.pdf).

Modern spammers create artificial Web landscapes of domains, servers, links, and pages to inflate the link scores of the targets they have been paid to promote. Spammers also engage in *cloaking*, the process of delivering different content to crawlers than to site visitors.

Search engine companies use manual and automated analysis of link patterns and content to identify spam sites that are then included in a blacklist. A crawler can reject links to URLs on the current blacklist and can reject or lower the priority of pages that are linked to or from blacklisted sites.

FINAL CRAWLING THOUGHTS

The full story of Web crawling must include decoding hyperlinks computed in JavaScript; extraction of indexable words, and perhaps links, from binary documents such as PDF and Microsoft Word files; and converting character encodings such as ASCII, Windows codepages, and Shift-JIS into Unicode for consistent indexing (www.unicode.org/standard/standard.html).

Engineering a Web-scale crawler is not for the unskilled or fainthearted. Crawlers are highly complex parallel systems, communicating with millions of different Web servers, among which can be found every conceivable failure mode, all manner of deliberate and accidental crawler traps, and every variety of noncompliance with published standards. Consequently, the authors of the Mercator crawler found it necessary to write their own versions of low-level system software to achieve required performance and reliability (www.research.compaq.com/SRC/mercator/papers/www/paper.html).

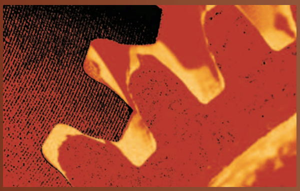
It is not uncommon to find that a crawler has locked up, ground to a halt, crashed, burned up an entire network traffic budget, or unintentionally inflicted a denial-of-service attack on a Web server whose operator is now very irate.

Part two of this two-part series (*Computer*, How Things Work, Aug. 2006) will explain how search engines index crawled data and how they processes queries.

David Hawking is a principal research scientist at CSIRO ICT Centre, Canberra, Australia, and Chief Scientist at funnelback.com. Contact him at david.hawking@csiro.au.

Web Search Engines: Part 2

David Hawking
CSIRO ICT Centre



A data processing "miracle" provides responses to hundreds of millions of Web searches each day.

Part 1 of this two-part series (How Things Work, June 2006, pp. 86-88) described search engine infrastructure and algorithms for crawling the Web. Part 2 reviews the algorithms and data structures required to index 400 terabytes of Web page text and deliver high-quality results in response to hundreds of millions of queries each day.

INDEXING ALGORITHMS

Search engines use an inverted file to rapidly identify indexing terms—the documents that contain a particular word or phrase (J. Zobel and A. Moffat, "Inverted Files for Text Search Engines," to be published in *ACM Computing Surveys*, 2006). An inverted file is a concatenation of the postings lists for each distinct term. In its simplest form, each postings list comprises a sorted list of the ID numbers of the documents that contain it. A fast-lookup term dictionary references the postings lists for each term.

An indexer can create an inverted file in two phases. In the first phase, *scanning*, the indexer scans the text of each input document. For each indexable term it encounters, the indexer writes a posting consisting of a document number and a term number to a temporary file. Because of the scanning process, this file will naturally be in document number order.

In the second phase, *inversion*, the indexer sorts the temporary file into term number order, with the document number as the secondary sort key. It also records the starting point and length of the lists for each entry in the term dictionary.

REAL INDEXERS

As Figure 1 shows, for high-quality rankings, real indexers store additional information in the postings, such as term frequency or positions. (For additional information, see S. Brin and L. Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine;" www-db.stanford.edu/pub/papers/google.pdf.)

Scaling up. The scale of the inversion problem for a Web-sized crawl is enormous: Estimating 500 terms in each of 20 billion pages, the temporary file might contain 10 trillion entries.

An obvious approach, document partitioning, divides up the URLs between machines in a cluster in the same way as the crawler. If the system uses 400 machines to index 20 billion pages, and the machines share the load evenly, then each machine manages a partition of 50 million pages.

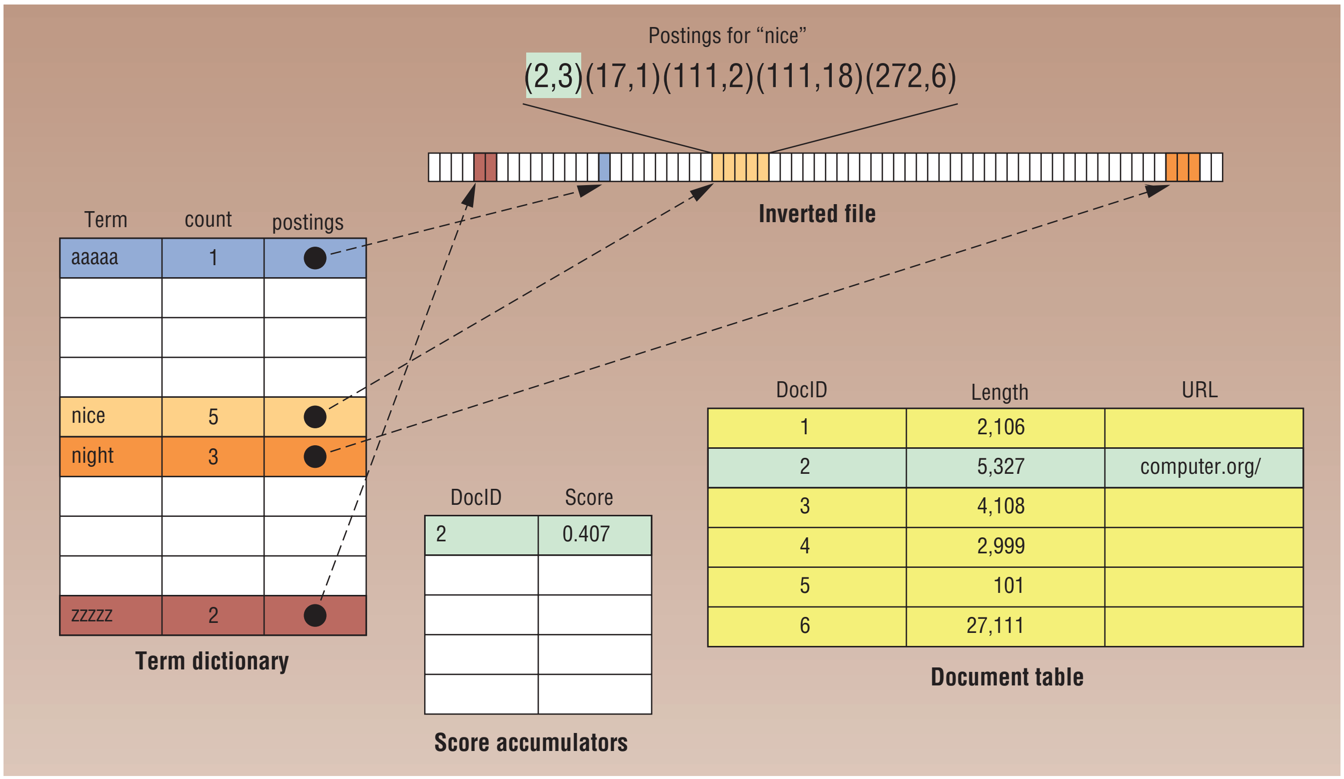
Even with 400-fold partitioning, each inverted file contains around 25 billion entries, still a significant indexing challenge. An efficient indexer builds a partial inverted "file" in main memory as it scans documents and stops when available memory is exhausted. At that point, the indexer writes a partial inverted file to disk, clears memory, and starts indexing the next document. It then repeats this process until it has scanned all documents. Finally, it merges all the partial inverted files.

Term lookup. The Web's vocabulary is unexpectedly large, containing hundreds of millions of distinct terms. How can this be, you ask, when even the largest English dictionary lists only about a million words? The answer is that the Web includes documents in all languages, and that human authors have an apparently limitless propensity to create new words such as acronyms, trademarks, e-mail addresses, and proper names. People certainly want to search for R2-D2 and C-3PO as well as IBM, B-52, and, yes, Yahoo! and Google. Many nondictionary words are of course misspellings and typographical errors, but there is no safe way to eliminate them.

Search engines can choose from various forms of tries, trees, and hash tables for efficient term lookup (D.E. Knuth, *The Art of Computer Programming: Sorting and Searching*, Addison-Wesley, 1973). They can use a two-level structure to reduce disk seeks, an important consideration because modern CPUs can execute many millions of instructions in the time taken for one seek.

Compression. Indexers can reduce demands on disk space and memory by using compression algorithms for key data structures. Compressed data structures mean fewer disk accesses and can lead to faster indexing and faster query processing, despite the CPU cost of compression and decompression.

Phrases. In principle, a query processor can correctly answer phrase queries such as "National Science Foundation" by intersecting postings lists containing word position infor-



Term	count	postings
aaaaa	1	●
nice	5	●
night	3	●
zzzzz	2	●

DocID	Score
2	0.407

DocID	Length	URL
1	2,106	
2	5,327	computer.org/
3	4,108	
4	2,999	
5	101	
6	27,111	

Figure 1. Inverted file index and associated data structures. In this simplified example, the alphabetically sorted term dictionary allows fast access to a word's postings list within the inverted file. Postings contain both a document number and a word position within the document. Note that "nice" occurs five times all told, twice in document 111. On the basis of the first posting, the query processor has calculated a relevance score for document 2.

mation. In practice, this is much too slow when postings lists are long.

Special indexing tricks permit a more rapid response. One trick is to precompute postings lists for common phrases. Another is to subdivide the postings list for a word into sublists, according to the word that follows the primary word. For example, postings for the word "electrical" might be divided into sublists for "electrical apparatus," "electrical current," "electrical engineering," and so on.

Anchor text. Web browsers highlight words in a Web page to indicate the presence of a link that users can click on. These words are known as *link anchor text*. Web search engines index anchor text with a link's target—as well as its source—because anchor text provides useful descriptions of the target, except "click here," of course. Pages that have many incoming links accumulate a variety of anchor text descriptions. The most useful descriptions can be repeated thousands of times, providing a strong signal of what the page is about. Anchor text contributes strongly to the quality of search results.

Link popularity score. Search engines assign pages a link popularity score derived from the frequency of incoming links. This can be a simple count or it can exclude links from within the same site. PageRank, a more sophisticated link popularity score, assigns different weights to links depending on the source's page rank. PageRank computation is an eigenvector calculation on the page-page link connectivity matrix.

Processing matrices of rank 20 billion is computationally impractical, and researchers have invested a great deal of brainpower in determining how to reduce the scale of PageRank-like problems. One approach is to compute HostRanks from the much smaller host-host connectivity matrix and distribute PageRanks to individual pages within each site afterwards. PageRank is a Google technology, but other engines use variants of this approach.

Query-independent score. Internally, search engines rank Web pages independently of any query, using a combination of query-independent factors such as link popularity, URL brevity, spam score, and perhaps the frequency with which users click them. A page with a high query-independent score has a higher a priori probability of retrieval than others that match the query equally well.

QUERY PROCESSING ALGORITHMS

By far the most common type of query that search engines receive consists of a small number of query words, without operators—for example, "Katrina" or "secretary of state." Several researchers have reported that the average query length is around 2.3 words.

By default, current search engines return only documents containing all the query words. To achieve this, a simple-query processor looks up each query term in the term dictionary and locates its postings list. The processor simultaneously scans the postings lists for all the terms to find documents in common. It stops once it has found the required number of matching documents or when it reaches the end of a list.

In a document-partitioned environment, each machine in a cluster must answer the query on its subset of Web pages and then send the top-ranked results to a coordinating machine for merging and presentation.

REAL QUERY PROCESSORS

The major problem with the simple-query processor is that it returns poor results. In response to the query "the Onion" (seeking the satirical newspaper site), pages about soup and gardening would almost certainly swamp the desired result.

Result quality

Result quality can be dramatically improved if the query processor scans to the end of the lists and then sorts the long list of results according to a relevance-scoring function that takes into account the number of query term occurrences, document length, inlink score, anchor text matches, phrase matches, and so on. The MSN search engine reportedly takes into account more than 300 ranking factors.

Unfortunately, this approach is too computationally expensive to allow processing queries like "the Onion" quickly enough. The postings list for "the" contains billions of entries, and the number of documents needing to be scored and sorted—those that contain both "the" and "onion"—is on the order of tens of millions.

Speeding things up

Real search engines use many techniques to speed things up.

Skipping. Scanning postings lists one entry at a time is very slow. If the next document containing "Onion" is number 2,000,000 and the current position in the "the" list is 1,500,000, the search obviously should skip half a million postings in the latter list as fast as possible. A small amount of additional structure in postings lists permits the query processor to skip forward in steps of hundreds or thousands to the required document number.

Early termination. The query processor can save a great deal of computation if the indexer creates indexes in which it sorts postings lists in order of decreasing value. It can usually stop processing after scanning only a small fraction of the lists because later results are less likely to be valuable than those already seen. At first glance, early termination seems to be inconsistent with skipping and compression techniques, which require postings to be in document number order. But there is a solution.

Clever assignment of document numbers. Instead of arbitrarily numbering documents, the crawler or indexer can number them to reflect their decreasing query-independent score. In other words, document number 1 (or 0 if you are a mathematician!) is the document with the highest a priori probability of retrieval.

This approach achieves a win-win-win solution: effective postings compression, skipping, and early termination.

Caching. There is a strong economic incentive for search engines to use caching to reduce the cost of answering queries. In the simplest case, the search engine precomputes and stores HTML results pages for thousands of the most popular queries. A dedicated machine can use a simple in-memory lookup to answer such queries. The normal query processor also uses caching to reduce the cost of accessing commonly needed parts of term dictionaries and inverted files.

Limited space prevents discussion of many other fascinating aspects of search engine operation, such as estimating the number of hits for a query when it has not been fully evaluated, generating advertisements targeted to the search query, searching images and videos, merging search results from other sources such as news, generating spelling suggestions from query logs, creating query-biased result snippets, and performing linguistic operations such as word-stemming in a multilingual environment.

A high priority for search engine operation is monitoring the search quality to ensure that it does not decrease when a new index is installed or when the search algorithm is modified. But that is a story in itself. ■

David Hawking is a principal research scientist at CSIRO ICT Centre, Canberra, Australia, and chief scientist at funnelback.com. Contact him at david.hawking@csiro.au.