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

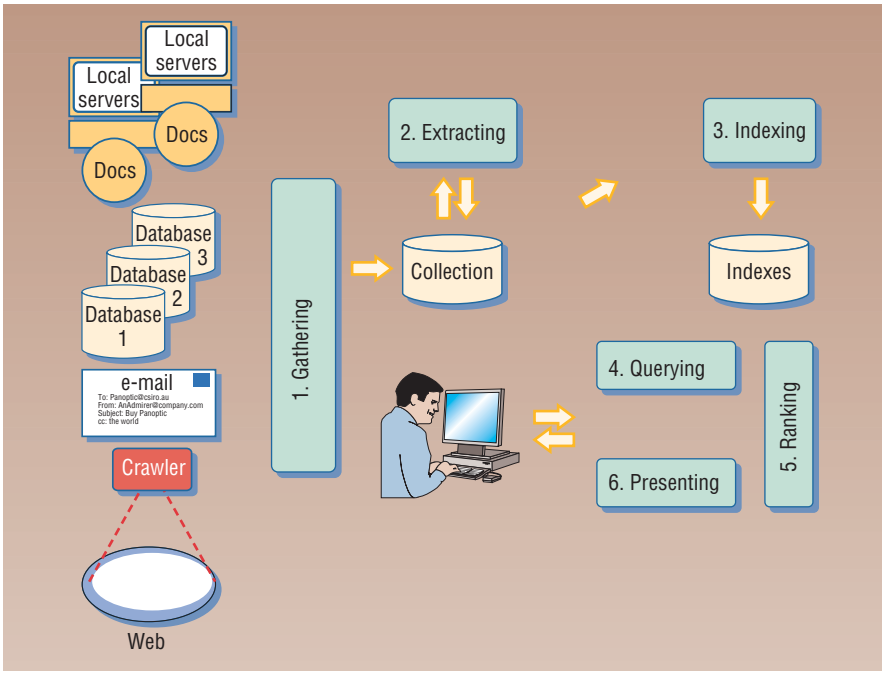

Figure 1. Generic search engine architecture. Enterprise search engines must provide adapters (top left) for all kinds of Web and non-Web data, but these are not required in a purely Web search.

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.*

Computer welcomes your submissions to this bimonthly column. For additional information, contact Alf Weaver, the column editor, at weaver@cs.virginia.edu.