MIKE CAFARELLA AND
DOUG CUTTING, NUTCH

Building Nutch: Open Source

Search engines are as critical to Internet use as any other part of the network infrastructure, but they differ from other components in two important ways. First, their internal workings are secret, unlike, say, the workings of the DNS (domain name system). Second, they hold political and cultural power, as users increasingly rely on them to navigate online content.

When so many rely on services whose internals are closely guarded, the possibilities for honest mistakes, let alone abuse, are worrisome. Further, keeping search-engine algorithms secret means that further advances in the area become *less* likely. Much relevant research is kept behind corporate walls, and useful methods remain largely unknown.

To address these problems, we started the Nutch software project, an open source search engine free for anyone to download, modify, and run, either as an internal intranet search engine or as a public Web search service. As you may have just read in Anna Patterson's "Why Writing Your Own Search Engine Is Hard," writing a search engine is not easy. As such, our article focuses on Nutch's technical challenges, but of course we hope Nutch will offer improvements in both the technical and social spheres. By enabling more people to run search engines, and by making the code open, we hope search algorithms will become as transparent as their importance demands.

TECHNICAL CHALLENGES

Much of the challenge in designing a search engine is making it scale. Writing a Web crawler that can download a handful of pages is straightforward, but writing one that can regularly download the Web's nearly 5 billion pages is much harder.

Further, a search engine must be able to process queries efficiently. Requirements vary widely with site popularity: a search engine may receive anywhere from less than one to hundreds of searches per second.

Finally, unlike many software projects, search engines

Search

A case study in writing
an open source search engine

can have high ongoing costs. They may require lots of hardware that consumes lots of Internet bandwidth and electricity. We discuss deployment costs in more detail in the next section, but for now it's helpful to keep in mind a few ideas:

- The cost of one part of the search engine scales with the size of the document collection. The collection might be very small when Nutch is searching a single intranet, but could be as large as the Web itself.
- Another part of the search engine scales with the size of the query load. Each query takes a certain amount of time to process and consumes some bandwidth.
- With these two factors in mind, we've designed a system that can easily distribute the work of both fetching and query processing over a set of standard machines.

Figure 1 shows the system's components.

WebDB. *WebDB* is a persistent custom database that tracks every known page and relevant link. It maintains a small set of facts about each, such as the last-crawled date. WebDB is meant to exist for a long time, across many months of operation.

Since WebDB knows when each link was last fetched, it can easily generate a set of *fetchlists*. These lists contain every URL we're interested in downloading. WebDB splits the overall workload into several lists, one for each fetcher process. URLs are distributed almost randomly; all the links for a single domain are fetched by the same process, so it can obey politeness constraints.

The *fetchers* consume the fetchlists and start downloading from the Internet. The fetchers are "polite," meaning they don't overload a single site with requests, and they observe the Robots Exclusion Protocol. (This allows Web-site owners to mark parts of the site as off-limits to automated clients such as our fetcher.) Otherwise, the fetcher blindly marches down the fetchlist, writing down the resulting downloaded text.

Fetchers output WebDB *updates* and *Web content*. The updates tell WebDB about pages that have appeared or disappeared since the last fetch attempt. The Web content is used to generate the searchable index that users will actually query.

Note that the WebDB-fetch cycle is designed to repeat forever, maintaining an up-to-date image of the Web graph.

Indexing and Querying. Once we have the Web content, Nutch can get ready to process queries. The *indexer* uses the content to generate an inverted index of all terms and all pages. We divide the document set into a set of *index* segments, each of which is fed to a single *searcher* process.

We can thus distribute the current set of index segments over an arbitrary number of searcher processes, allowing us to scale easily with the query load. Further, we can copy an index segment to multiple machines and run a searcher over each one; that allows more good scaling behavior and reliability in case one or more of the searcher machines fail.

Each searcher also draws upon the Web content from earlier, so it can provide a cached copy of any Web page.

Finally, a pool of *Web servers* handle interactions with users and contact the searchers for results. Each Web server interacts with many different searchers to learn about the entire document set. In this way, the Web server is simultaneously acting as an HTTP server and a Nutch-search client.

Web servers contain very little state and can be easily reproduced to handle increased load. They need to be told only about the existing pool of searcher machines. The only state they do maintain is a list of which searcher processes are available at any time; if a given segment's searcher fails, the Web server will query a different one instead.

Quality. Generating high-quality results, of course, is the most important barrier for Nutch to overcome. If it cannot find relevant pages as well as commercial engines do, Nutch isn't much use. But how can it ever compete with large, paid engineering staffs?

- First, we believe high-quality search is a slowing target. By some measures of quality, the gap between the best search engine and its competitors has narrowed considerably. After several years of intense focus on search results, anecdotal evidence suggests gains in quality are harder to find. The everyday search user will find lots of new features on the various engines, but real differences in results quality are close to imperceptible.
- Second, although much search work takes place behind corporate walls, there is still a fair amount of public academic work. Many of the techniques that search engines use were discovered by IR (information retrieval) researchers in the 1970s. Some people have tried to tie IR in with advances in language understanding. With the advent of the Web, many different groups experimented with link-driven methods. We think there should be more public research, but there is already a good amount to draw upon.

- Third, we expect that Nutch will be able to incorporate academic advances faster than any other engine can. We think researchers and engineers will find Nutch very appealing. If it becomes the easiest platform for researchers to experiment on, taking advantage of the results should be extremely simple.
- Finally, we'll rely on the traditional advantages of open source projects. More people from more places should work on Nutch, which means faster bug finding, more ideas, and better implementations. In the long term, a worldwide shared effort supported by research at a number of institutions should eventually be able to surpass the private efforts of any company.

Once an open source search solution is as good or better than proprietary implementations, there should be little reason for companies to use anything but the open source version. It will be cheaper to maintain and work as well.

Spam. A high search ranking can be extremely valuable to a Web-site owner—so valuable that many sites try to "spam" search engines with specially formulated content in an effort to raise their rankings. As with e-mail spam, the spammer can benefit at a heavy cost to everyday users.

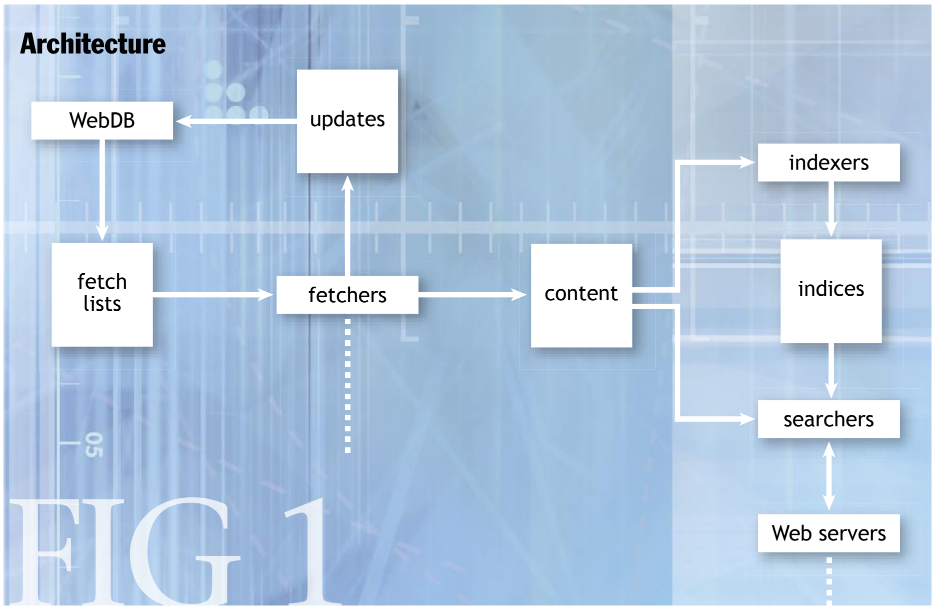
How does this work in practice? Search engines tend to use a well-known set of guidelines to measure a page's relevance to a given query. For example, all other things being equal, a page that contains the word *parrot* 10 times is more about parrots than a page that has the word just once. A page with lots of incoming links from other sites is more important than a page with fewer incoming links.

That means it can be fairly easy to trick a naive search engine. Want to make sure every parrot lover finds your page? Repeat the word *parrot* 600 times somewhere on your page. Want to raise your page's in-link count? Pay a type of site known as a "link farm" to add thousands of links aimed at your page.

Of course, the consequence is that search results can become choked with sites that are not truly relevant, but have "gamed" the system successfully. Good search engines don't want their results to become useless, so they do everything possible to detect these spam tricks. Spammers, in turn, modify their tricks to avoid detection. The result is an arms race between search engine and spammer.

Here are some well-known spam techniques, along with methods to defeat them:

- Web sites write documents that contains long repetitions of certain words. *Search engines counter by eliminating terms that appear consecutively more than a certain number of times.*
- Web sites do the same trick, but intersperse the repeated term along with good-looking intervening text. *Search engines counter by checking whether the statistical distribution of the words in the document matches the typical English-language profile. If it's too far afield, the site is marked as a spammer.*
- Web sites that want high rankings regardless of query put spurious "invisible" text on the page. Say the site offers a page about electronics, all rendered on a white background. The very same page might contain a long essay about, say, Britney Spears, all rendered in white text. Users won't see it, but the search engine will. *Search engines counter by computing the visible portion of the HTML and tossing the rest, or even by penalizing pages that use any invisible text.*
- Web sites use the "User-Agent" tag to identify the type of browser. If the browser is a piece of desktop software, the Web site returns regular content. If the browser is a crawler for a search engine, the Web site returns different content that contains thousands of repetitions of parrot. *Search engines fight against this by penalizing*



sites that give substantially different content for different browser types.

- Web sites use link farms to add to incoming link count. *Search engines find link farms by looking for statistically unusual link structures. The link farms are thrown away before computing link counts. Pages that participate in the farm may also be penalized.*

Some of these methods may rely on secrecy for their effectiveness, so some people ask how an open source engine could possibly handle spam. With full disclosure of code, won't a search engine lose the fight?

It's true that Nutch code won't hold any secrets. But these secrets are brittle anyway—spammers don't take long to defeat the latest defense. If search has to rely on secrecy to beat spam, the spammers will probably win.

In the world of e-mail spam, at least, the days of simple methods to defeat spammers seem to be over. Many of the latest techniques to defeat e-mail spam are statistics driven. With such methods, even intimate knowledge of the source code may not help spammers much. Although people may be reluctant to use such probabilistic spam detectors on e-mail for fear of deleting a single good message, the massive redundancy of Web information means false positives are not so great a tragedy.

Alternatively, the answer may lie in an analogy to cryptography. It has taken a long time for people to learn the counterintuitive notion that the most secure cryptographic systems are those that have the most public scrutiny. Most people who look at these systems are well motivated and work to improve them rather than to defeat them. They find problems before they can be exploited.

The analogy may be flawed, but it can't be tested without transparency. Nutch is currently the best shot at enabling some form of public review for defeating search engine spam.

DEPLOYMENTS/OPERATIONS

Scalability/Cost Effectiveness. One objection the Nutch project often hears is that search is simply too resource-hungry to be a good open source project. In fact, a Web search engine can be operated for fairly modest sums.

A note on index size: Web search engines make claims about the sizes of their indexes, but these are not directly comparable. Some count the number of pages they've fetched; some count the number of URLs that may be returned, even though they've not been fetched but only referenced by another page. Also, many pages are duplicates: a given site may respond to more than one host name, giving all of its pages multiple URLs. And although bigger is almost always better, it may not be much better. An index with just 100 million pages can perhaps satisfy 99 percent of users' searches as well as a 5-billion-page index. So if you are primarily interested in cost-effective usability, you might build only a 100-million-page system. But if you are interested in bragging rights and satisfying rare, obscure searches, then a larger index may be justified.

Here we will outline Nutch's operational costs. All figures are meant to be illustrative, since the performance and cost of hardware, software, and bandwidth are all changing.

Nutch deployments use two classes of machines: back-end machines, for crawling, database, link analysis, and indexing tasks; and front-end machines, which perform searches and serve search results.

A typical back-end machine is a single-processor box with 1 gigabyte of RAM, a RAID controller, and eight hard drives. The filesystem is mirrored (RAID level 1) and provides 1 terabyte of reliable storage. Such a machine can be assembled for a cost of about $3,000.

One such back-end machine is required for every 100 million pages. Thus, to maintain an index of 1 billion pages requires 10 back-end machines, or about $30,000 in hardware.

A typical front-end machine is a single-processor box with 4 gigabytes of RAM and a single hard drive. Such a machine can be assembled for about $1,000.

The query-handling capacity of front-end machines varies, depending on how much each must search. For example, if each front-end machine is given 25 million pages to search, then each can perform about two searches per second. Thus, a 100-million-page index could be searched with four front-end machines ($4,000) while a 1-billion-page index requires 40 front-end machines ($40,000), but such configurations could still handle only two searches per second. In this case, access to a disk-resident index is the primary bottleneck.

Query handling is more cost effective when primary index structures fit within RAM. In particular, if each front-end machine is required to handle only 2 million pages, then each can handle perhaps 50 searches per second. In this configuration a 100-million-page index would require 50 front-end machines ($50,000) and a 1-billion-page index would require 500 machines

($500,000). This is half the cost per query of the first case. Here the bottleneck is primarily the CPU. Further search software optimizations can make this configuration even more cost effective.

Note that as traffic increases, front-end hardware quickly becomes the dominant hardware cost.

Thus far we have discussed only the raw hardware costs. In addition, there are hosting costs. These are primarily electricity (as consumed both directly by the hardware and by the air conditioning required to cool the hardware), bandwidth, and others (racks, network equipment, facility rental, etc.). Electricity dominates these costs, and together, these costs easily dominate raw hardware costs. For example, you might amortize the cost of hardware over three years, so that $100,000 of hardware is less than $3,000 per month; but power, space, and bandwidth for 100 machines can easily cost more than that. Since hosting costs are even more variable than hardware prices, let's just assume that hosting costs are approximately the same as three-year amortized hardware costs. Thus, a complete system might cost anywhere between $800 per month for two-search-per-second performance over 100 million pages, to $30,000 per month for 50-page-per-second performance over 1 billion pages.

A note on bandwidth consumption: If we assume that Web pages average around 10 kilobytes, and each must be re-fetched monthly, then fetching 1 billion pages per month requires around 40 Mbps (megabits per second) inbound. Bandwidth costs are typically symmetric, so if you have purchased 40 Mbps inbound, you have also purchased 40 Mbps outbound. If we further assume that a search result page is also around 10 KB, then, until search traffic surpasses 400 searches per second, in a 1-billion-page system, the crawler's needs dominate bandwidth; after that, search dominates.

In summary, although it's true that Google and Yahoo probably cost a lot of money to operate, many Web search engines may not serve nearly as much traffic and need not search nearly so many pages. In a world with lots of deployed search engines, the vast majority will serve small audiences. The costs are also well within reach for research groups, governmental departments, and small- to medium-size companies.

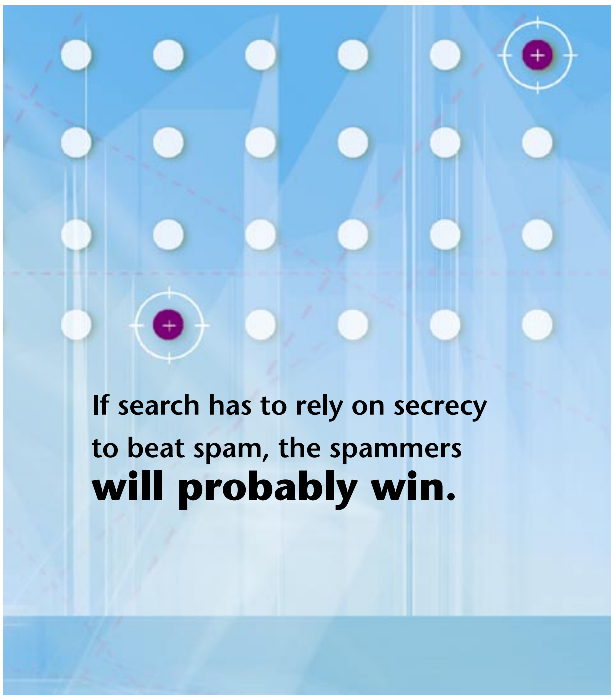
One much trickier source of cost savings is automating most system administration tasks. We believe there is a lot of ground to be gained here, and Nutch has not yet started. It's not clear how to use the open source programming style for something that's so tied to the deployment, but we need to do it.

Who Should Run Nutch-Based Web Search Engine(s)?
Nutch.org is dedicated to making the Nutch software better for everyone. That might mean running a small demo site or making a search service available for academic research, but we do not intend to run a destination search site. Running such a service would put Nutch in competition with its users. Instead, we hope that primarily other institutions will run the Nutch software.

Governments, universities, and nonprofits are terrific candidates for Nutch. These organizations often have special obligations that for-profit companies don't (e.g., a seniors' organization might want to offer search with a special usability focus), so having the source code to Nutch is a huge advantage. Further, these groups often don't have lots of cash to spend on solutions.

We don't have great data yet on who is running Nutch. As far as we can tell, the most active Nutch users are universities and academic research groups. Some are using Nutch as part of a class, and some are using it because their research depends on access to indexed pages that they can control. Others are pulling apart the system, taking elements that seem useful. It's too early to expect any updates back from researchers, but we hope this is coming soon.

One type of nonprofit in particular that we hope to see is a PSE (public search engine), a search site that is as usable as any commercial one, but that operates without advertising or commercial engagement. These engines will help make good on Nutch's



If search has to rely on secrecy to beat spam, the spammers **will probably win.**

promise to make search results more transparent to users. Conversely, they will make for-profit engines easier to spot if they adjust rankings for commercial gain.

A PSE might get its funds through donations from users, corporations, or foundations, just as public broadcasting channels do. It's worth noting that PSEs do not need to process a huge percentage of search queries to be successful. Their existence will help ensure that search users always have a good alternative (one that doesn't exist today).

What about for-profit corporations? We think lots of companies will want to run small search engines for in-house use or on their public Web sites. For most of these companies, search will be just another item they have to take care of, not their main focus.

Nutch should also enable small search-technology companies to be more creative, just as other open source projects have enlarged what small teams can accomplish.

We hope that Nutch, by providing free, open source Web search software, will help both to promote transparency in Web search and to advance public knowledge of Web-search algorithms. Q

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums

MIKE CAFARELLA worked as a software engineer at Silicon Valley startups Marimba Corporation and Tellme Networks from 1998 to 2002. In 2002 he began work on the Nutch project, which he continues. In 2003 he started a Ph.D. program in computer science at the University of Washington. He graduated from Brown University in 1996. In 1997, he earned a M.A. from the University of Edinburgh while on a Fulbright scholarship to the United Kingdom.

DOUG CUTTING has worked on search technology for more than 15 years. This includes five years at Xerox PARC, three years at Apple with its Advanced Technology Group, and more than four years at Excite. In 1998 he wrote Lucene (http://jakarta.apache.org/lucene/), an open source search library that subsequently became part of the Apache Jakarta project. In 2002 he started Nutch (http://www.nutch.org/), an open source Web search application.

dtSearch® Instantly Search Gigabytes of Text Across a PC, Network, Intranet or Internet Site

Publish Large Document Collections to the Web or to CD/DVD

- over two dozen indexed, unindexed, fielded & full-text search options
- highlights hits in HTML, XML & PDF while displaying embedded links, formatting & images
- converts other file types (word processor, database, spreadsheet, email, ZIP, Unicode, etc.) to HTML for display with highlighted hits

"The most powerful document search tool on the market" –*Wired Magazine*

"Intuitive and austere ... a superb search tool" –*PC World*

"Blindingly fast" –*Computer Forensics: Incident Response Essentials*

"A powerful arsenal of search tools" –*The New York Times*

dtSearch "covers all data sources ... powerful Web-based engines" –*eWEEK*

"Searches at blazing speeds" –*Computer Reseller News Test Center*

In the past two years, over half of the Fortune 15 purchased dtSearch *developer* or *network* licenses.

1-800-IT-FINDS
sales@dtsearch.com

See www.dtsearch.com for:
- hundreds of developer case studies & reviews
- fully-functional evaluations

dtSearch DESKTOP with Spider
"Industrial-strength ... superb" –PC Magazine
◇ $199
dtSearch WEB with Spider
"Industrial-strength ... superb" –PC Magazine
◇ $999 per server
dtSearch Text Retrieval ENGINE
"Industrial-strength ... superb" –PC Magazine
◇ for Win & .NET
◇ for Linux
◇ call for pricing
dtSearch PUBLISH for CD/DVDs
"Industrial-strength ... superb" –PC Magazine
◆ from $2,500
dtSearch NETWORK with Spider
"Industrial-strength ... superb" –PC Magazine
◆ from $800

The Smart Choice for Text Retrieval® since 1991