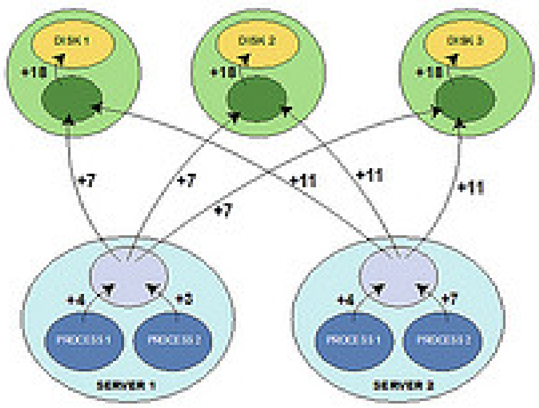
The Anatomy of Search Technology: Crawling using Combinators

Monday, May 28, 2012 at 9:15AM

Todd Hoff in Example

*This is the second guest post (*part 1*,* part 3*) of a series by Greg Lindahl, CTO of blekko, the spam free search engine. Previously, Greg was Founder and Distinguished Engineer at PathScale, at which he was the architect of the InfiniPath low-latency InfiniBand HCA, used to build tightly-coupled supercomputing clusters.*



What's so hard about crawling the web?

Web crawlers have been around as long as the Web has -- and before the web, there were crawlers for gopher and ftp. You would think that 25 years of experience would render crawling a solved problem, but the vast growth of the web and new inventions in the technology of webspam and other unsavory content results in a constant supply of new challenges. The general difficulty of tightly-coupled parallel programming also rears its head, as the web has scaled from millions to 100s of billions of pages.

Existing Open-Source Crawlers and Crawls

This article is mainly going to discuss blekko's crawler and its use of combinators, but if you'd like some more general introductions to the hard parts of crawling, I'd suggest looking at:

- The chapter on crawling from the book Introduction to Information Retrieval
- The Apache Nutch open-source crawler
- The open source crawler heritrix from
- The 5-billion-page Common Crawl

Directed crawling

The most important feature of a crawler intending to not crawl the entire web is the ability to crawl only the most important pages. It's rumored that Google's web crawl and index is over 100 billion webpages, and Google announced in 2008 that their "crawl frontier"-- the list of all of the urls that they had seen on other webpages-- was over.

blekko knew that we only wanted to index and serve results for an index of just a few billion webpages. That's only a fraction of a percent of the webpages in Google's crawl frontier, so we need to be very good at crawling the best pages, and only the best pages. One way to do this is to compute the ranks of webpages as we crawl, including the ranks of the pages we haven't crawled yet.

The rank of a page is dependent upon the number and quality of incoming links, plus many other on-page measures, such as the text on the page, the number of ads compared to the amount of text, and so forth. The on-page measures can't be known before the page is first crawled, but the incoming links are known, from crawling other pages.

Using incoming links to rank a webpage is, of course, something which is already well-gamed by a lot of Internet spammers. Some of these bogus links come from other spammer websites, and some come from legitimate websites with reasonable content. I have a bunch of old, highly-ranked webpages on various topics, and I receive an endless trickle of "link trading" emails, mostly sent by software packages which automate the link trading game. Finding and ignoring bad links from legitimate websites is much more difficult, and often can't be done until many linked pages have been fully crawled.

Combinators

Now let's see how **combinators** (which we discussed in the previous blog posting) might make doing some of these computations easier.

First, we have many things which we'd like to make unique counts of: the geographic diversity of incoming links, the network diversity of incoming links, and so forth. A page with a lot of incoming links is less interesting if all of the incoming links come from the same class-C IP network. We use the **logcount** combinator to count these quantities efficiently (in both time and space -- 16 bytes per count), without double-counting anything as we crawl and recrawl the web. The down-side of using logcount is that the counts are approximate; for some important quantities, we choose variants of logcount which require as many as 256 bytes of state, in order to have a better approximation of the exact answer.

Next, we frequently need to manipulate the lists of outgoing and incoming links to a webpage. In most relational databases, this data is usually represented by a series of rows in a table, and we would fetch this data by asking for all records where the destination of the link equaled a particular URL. This is an expensive operation, and since the number of inbound links can be large (millions, in many cases), we would need some sort of way of getting rid of less important (lower-ranked) rows in this table, in order to keep the table size reasonable.

The **TopN** combinator solves both of these issues. As a finite-sized list, it can be read in single operation, and it is self-trimming in size. As an example of why we might want to manipulate this list of incoming webpages at crawl time, consider the fact that traded or purchased links often have identical anchortext. By examining the incoming anchortext before crawling a page, we can avoid crawling it at all. An index-time check could discover the same similarity of anchortext, but that would be too late to avoid wasting resources crawling it.

In addition to url-level information, we also keep host-level summaries of what the crawler has learned; for example, we have a TopN summary and counts of host-to-host links. This summary is useful for discovering groups of hosts with large numbers of in-group links; we use this data to discount the value of these links.

All that other stuff

In addition to what we've discussed already -- finding outgoing links, and computing the ranks of uncrawled pages -- blekko's crawler does quite a bit of other work. If a date is found on a webpage, the crawler immediately sends the page to be indexed for the additional indexes which support blekko's /date and /daterange features (see this page about blekko's advanced features for more details.) The crawler also immediately updates a bunch of lists such as the list of domains using a given IP address, the list of domains using a given analytics or advertising ID, and the lists of checksums which drive our duplicate text discovery system.

Crawling experiences

We learned a few lessons along the way. One important lesson is that it's critical to have an email address that webmasters can use to contact us privately if there's a crawler problem. We've fixed several bugs thanks to this. The most surprising was that webmasters (and the major crawlers) don't strictly follow the robots.txt specification, and expect blank lines in their robots.txt to have no effect. We also found that a significant fraction of websites, including many US government websites, only allow a small whitelist of crawlers to crawl their pages. Many of these websites are small, and there's no visible way to contact their webmasters to ask to be added to the whitelist.

Future directions

In the future, there is one major thing we'd like to add to our crawling system: is the ability to execute javascript. More and more of the web is hidden behind javascript, and while webmasters are somewhat careful to not hide their content where it can't be seen by most search engines, many webmasters do have an incentive to hide their analytics and advertising IDs so that they aren't as visible.

Related Articles

- On Hacker News
- The Anatomy Of Search Technology: Blekko’s NoSQL Database
