Web Crawling


Web Crawling

$S = \{\text{startpage}\}$

repeat
    remove an element $s$ from $S$
    foreach $(s, v)$
        if $v$ not crawled before
            insert $v$ in $S$
Issues

Theoretical:

- Startset $S$
- Choice of $s$ (crawl strategy)
- Refreshing of changing pages.

Practical:

- Load balancing (own resources and resources of crawled sites)
- Size of data (compact representations)
- Performance (I/Os).
Crawl Strategy

- Breath First Search
- Depth First Search
- Random
- Priority Search

Possible priorities:

- Often changing pages (how to estimate change rate?).
- Using global ranking scheme for queries (e.g. PageRank).
- Using query dependent ranking scheme for queries ("focused crawling", "collection building").
BFS is Good

Figure 1: Average PageRank score by day of crawl

Figure 2: Average day on which the top $N$ pages were crawled

[From: Najork and Wiener, 2001]

Statistics for crawl of 328 million pages.
PageRank Priority is Even Better

(but computationally expensive to use . . . )

Figure 2: The performance of various ordering metrics for $IB(P)$; $G = 100$

[From: Arasu et al., 2001]

Statistics for crawl of 225,000 pages at Stanford.
Load Balancing

Own resources:

- Bandwidth (control global rate of requests)
- Storage (compact representations, compression)
- Industrial-strength crawlers must be distributed (e.g. partition the url-space)
Load Balancing

Own resources:

- Bandwidth (control global rate of requests)
- Storage (compact representations, compression)
- Industrial-strength crawlers must be distributed (e.g. partition the url-space)

Resources of others:

- BANDWIDTH. Control local rate of requests (e.g. 30 sec. between request to same site).
- Identify yourself in request. Give contact info.
- Monitor the crawl.
- Obey the Robots Exclusion Protocol ([www.robotstxt.org](http://www.robotstxt.org)). [Also read the other material there.]
Efficiency

- RAM: never enough for serious crawls. Efficient use of disk based storage important. I/O when accessing data structures is often a bottleneck.

- CPU cycles: not a problem (Java and scripting languages are fine).

- DNS lookup can be a bottleneck (as normally synchronized). Asynchronous DNS: check GNU adns library.

Rates reported for serious crawlers: 200-400 pages/sec.
Example: Mercator

Figure 1: Mercator’s main components.

[From: Najork and Heydon, 2001]
Mercator

Further ideas:

- Fingerprinting ((sparse) hashfunction on strings).
- Continuous crawling—crawled pages put back in queue (prioritized using update history).
- Checkpointing (crash recovery).
- Very modular structure.
Details: Politeness

Figure 3: Our best URL frontier implementation

[From: Najork and Heydon, 2001]
Details: Efficient URL Elimination

- **Fingerprinting**
- Sorted file of fingerprints of seen URLs.
- Cache most used URLs.
- Non-cached URLs checked in batches (merge with file I/O).

<table>
<thead>
<tr>
<th>FP cache</th>
<th>Front-buffer containing FPs and URL indices</th>
</tr>
</thead>
<tbody>
<tr>
<td>025f978 038f7c97 05117c6f ...</td>
<td>035f4ca8 1 <a href="http://u.gov/gw">http://u.gov/gw</a></td>
</tr>
<tr>
<td></td>
<td>07f6de43 2 <a href="http://a.com/xz">http://a.com/xz</a></td>
</tr>
<tr>
<td></td>
<td>15f7885 3 <a href="http://w.org/gw">http://w.org/gw</a></td>
</tr>
<tr>
<td></td>
<td>23467676 4 <a href="http://b.net/hz">http://b.net/hz</a></td>
</tr>
<tr>
<td></td>
<td>27cc67ed 5 <a href="http://j.edu/tx">http://j.edu/tx</a></td>
</tr>
<tr>
<td></td>
<td>2f4e6710 6 <a href="http://m.mil/gd">http://m.mil/gd</a></td>
</tr>
<tr>
<td></td>
<td>327849c8 7 <a href="http://n.mil/gd">http://n.mil/gd</a></td>
</tr>
<tr>
<td></td>
<td>40678544 8 <a href="http://o.net/hi">http://o.net/hi</a></td>
</tr>
<tr>
<td></td>
<td>42ca6ff7 9 <a href="http://p.tw/ch">http://p.tw/ch</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Disk file containing URLs (one per front-buffer entry)</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://x.com/hr">http://x.com/hr</a></td>
</tr>
<tr>
<td><a href="http://y.org/ox">http://y.org/ox</a></td>
</tr>
<tr>
<td><a href="http://z.org/yz">http://z.org/yz</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FP disk file</th>
<th>Front-buffer containing FPs and URL indices</th>
</tr>
</thead>
<tbody>
<tr>
<td>025fe427 04ff5234 07852310 ...</td>
<td>035f4ca8 1 <a href="http://u.gov/gw">http://u.gov/gw</a></td>
</tr>
<tr>
<td></td>
<td>07f6de43 2 <a href="http://a.com/xz">http://a.com/xz</a></td>
</tr>
<tr>
<td></td>
<td>15f7885 3 <a href="http://w.org/gw">http://w.org/gw</a></td>
</tr>
<tr>
<td></td>
<td>23467676 4 <a href="http://b.net/hz">http://b.net/hz</a></td>
</tr>
<tr>
<td></td>
<td>27cc67ed 5 <a href="http://j.edu/tx">http://j.edu/tx</a></td>
</tr>
<tr>
<td></td>
<td>2f4e6710 6 <a href="http://m.mil/gd">http://m.mil/gd</a></td>
</tr>
<tr>
<td></td>
<td>327849c8 7 <a href="http://n.mil/gd">http://n.mil/gd</a></td>
</tr>
<tr>
<td></td>
<td>40678544 8 <a href="http://o.net/hi">http://o.net/hi</a></td>
</tr>
<tr>
<td></td>
<td>42ca6ff7 9 <a href="http://p.tw/ch">http://p.tw/ch</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Disk file containing URLs (one per back-buffer entry)</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://x.com/hr">http://x.com/hr</a></td>
</tr>
<tr>
<td><a href="http://y.org/ox">http://y.org/ox</a></td>
</tr>
<tr>
<td><a href="http://z.org/yz">http://z.org/yz</a></td>
</tr>
</tbody>
</table>

**Figure 4: Our most efficient disk-based DUE implementation**

[From: Najork and Heydon, 2001]
Some Experiences

Figure 6: Outcome of download attempts

Figure 7: Distribution of content types

Figure 8: Distribution of document sizes
Some Experiences

Figure 8: Distribution of document sizes

Pages (of type text/html) account for nearly two-thirds of all documents; images (in both GIF and JPEG formats) account for another 30%; all other content types combined account for less than 5%.

Figure 9a shows the distribution of pages over web servers, while Figure 9b measures content in bytes. Both figures are plotted on a log-log scale, and in both, a point \( x; y \) indicates that \( x \) web servers had at least \( y \) pages/bytes. The near-linear shape of the plot in Figure 9a indicates that the distribution of pages over web servers is Zipfian.

Figure 10 shows the distributions of web servers and web pages across top-level domains. About half of the servers and pages fall into the .com domain. For the most part, the numbers of hosts and pages in a top-level domain are well-correlated. However, there are some interesting wrinkles. For example, the .edu domain contains only about 1.53% of the hosts, but 5.56% of the total pages. In other words, the average university web server contains almost four times as many pages as the average server on the web at large.

5 Conclusion

High-performance web crawlers are an important component of many web services. Building a high-performance crawler is a non-trivial endeavor: the data manipulated by the crawler is too big to fit entirely in memory, so there are performance issues related to how to balance the use of disk and memory. This chapter has enumerated the main components required in any crawler, and it has discussed design alternatives for some of those components. In particular, the chapter described Mercator, an extensible, distributed, high-performance crawler written entirely in Java.

Mercator's design features a crawler core for handling the main crawling tasks, and extensibility through a component-based architecture that allows users to supply new modules at run-time for performing customized crawling tasks. These extensibility features have been quite successful. We were able to adapt Mercator to a variety of crawling tasks, and the new code was typically quite small (tens to hundreds of lines). Moreover, the flexibility afforded by the component model encouraged us to experiment with different implementations of the same functional components, and thus enabled us to discover new and efficient data structures. In our experience, these innovations produced larger performance gains than low-level tuning of our user-space code [13].

Mercator's scalability design has also worked well. It is easy to configure the crawler for varying memory footprints. For example, we have run it on machines...

[From: Najork and Heydon, 2001]
Further Resources

Further resources for implementing a crawler:

- Another good paper with practical info:
  Shkapenyuk and Suel: *Design and Implementation of a High-Performance Distributed Web Crawler*. IEEE Int. Conf. on Data Engineering (ICDE), February 2002. (http://cis.poly.edu/suel/papers/crawl.ps)

- HTML specification (www.w3.org)

- A free book on programming web agents.
  (http://www.oreilly.com/openbook/webclient)

- Software libraries (Java, Perl, Python, C++) for net programming.