Building a search engine with Clojure

Filip de Waard (fmw@vixu.com)

Code: https://github.com/fmw/alida

What does a search engine need to do?

- data retrieval (crawling),
- data storage (optional),
- data extraction (scraping),
- indexing,
- search,
- presentation.

Popular open source crawlers

Java:

- Apache Nutch: http://nutch.apache.org/
- Heritrix: https://github.com/internetarchive/heritrix3

Python:

- Scrapy: http://scrapy.org/
- Mechanize: http://wwwsearch.sourceforge.net/mechanize/

Other:

- wget: http://www.gnu.org/software/wget/
- HTTrack: http://www.httrack.com/page/1/en/index.html

Web crawling process

1. Download a seed page,
2. store page (optional),
3. extract links,
4. filter links,
5. download next page,
6. determine relevance,
7. repeat the steps for every relevant link.

Crawling approaches

Brute force:

- crawl and process everything you can get your hands on.

Directed crawling:

- specifically tell the crawler where to go.

Weighted crawling:

- start the crawler with a seed URI,
- run a scoring function over every new page,
- use the score to decide how to continue.

Roll your own crawler in Clojure

Leverage existing libraries:

- clj-http (wrapper for Apache HttpComponents),
- Enlive (selector-based data extraction & templating),
- Jsoup (Java HTML parser),
- Apache Lucene (search and indexing library written in Java).

Prototype:

- CouchDB through Clutch.

Recommended for production:

- Apache Hadoop for distributed crawling,
- Apache HBase and/or HDFS for storage.

Crawling considerations

Being polite:

- respect robots.txt,
- delay between requests.

Dealing with anti-crawling measures and non-standard content:

- headless browser for dynamic content (Selenium has good Clojure support),
- use OCR for images (e.g. tesseract-ocr),
- be aware of blocking through the user-agent header,
- don't get trapped in a loop,
- use a service like Amazon EC2 to recycle IP addresses.

Directed crawl

- Follow all links that start with "/en" on the seed page,
- only follow the external links in the content div on those pages,
- there is no :next key for that level, so the crawl stops there.

```
@(directed-crawl "external-links"
                 1000 ;; be nice and wait 1 second between requests
                 "http://www.vixu.com/"
                 [{:selector [:a]
                   :path-filter #"^/en.*"
                   :next [{:selector [[:div#content] [:a]]
                           :filter #"^(http://www.vixu.com/.*){0}.*"}]}]))
```

Weighted crawling scoring functions

Example page-scoring-fn that counts the number of occurrences of the string "website-management" on every requested page:

```
(defn page-scoring-fn [uri request]
  (count (re-seq #"website-management" (:body request))))
```

Example link-checker-fn that limits the crawl to Vixu.com:

```
(defn link-checker-fn [uri]
  (not (nil? (re-matches #"^http://www.vixu.com/.*" uri))))
```

Weighted crawl function

- recursively follows links,
- crawls external links in a new thread,
- saves the page to the database if the result from the page-scoring-fn is positive,
- doesn't follow links from a page when the page-scoring-fn returns a negative value,
- only follows links if calling link-checker-fn on them returns true.

```
@(weighted-crawl "alida" ;; database
                 "website-management-crawl"
                 1000 ;; delay in ms
                 "http://www.vixu.com/" ;; seed-uri
                 page-scoring-fn
                 link-checker-fn)
```