Concurrent Thread-based Web Crawler

Initial Plan

One Semester Individual Project
CM3202
40 Credits

Author: Jack Parsons
C1327650

Supervised by: David W Walker

Moderated by: Paul Rosin

Project Description

Crawling the web has been an interesting problem since the birth of the internet. The more efficient the web crawler then the more pages you may crawl over a period of time. Due to the ever expanding nature of the internet, this places large demands on the current web crawlers used to index this huge amount of data. As of July 2008, Google announced that its own crawlers had found one trillion unique web addresses.[1] With such a huge amount of data to process, it shows a need for a fast and scalable web crawler.
This project aims to create a web crawler, similar in functionality to GNU Wget's[2] spider functionality. It should take a list of web addresses as an input, and output links not found in the original list to the command line or to an output file. The crawler must provide access to command line flags to influence or constrain the search, for example to limit the search to a specific domain or depth. The software created should be concurrent and highly scalable, or as close to perfectly scalable as possible. The web crawler will be created in the programming language Haskell, in an attempt to maximise scalability and ease the creation of non deterministic code. The resulting program should adhere to politeness policies and must avoid any online 'traps' for web crawlers.

Project Aims and Objectives

The web crawler must attempt to maximise the use of the bandwidth provided by utilising HTTP compression[3] and other methods.

- Where viable, the web crawler must attempt to maximise the possible usage of bandwidth, for example downloading a mobile site instead of the desktop site, only if the number of links it provides is equal to that of the desktop site. The method which obtains the most links will be used. HTTP compression will also be utilised where possible to further this aim. Research must be done in this area to see whether the bandwidth saved will justify the processing cost of decompressing the content.

The web crawler must ensure that no web page is visited more than once.

- This contributes to the above aim; by ensuring no web page is visited more than once, no bandwidth is wasted. To achieve this, an algorithm such as multiple path pruning or cycle checking must be used. Tests will be carried out in this area to choose an appropriate algorithm to mitigate this.

The web crawler should respect the robots exclusion standard.[4]

- The robots.txt of a site must be parsed before a site is crawled; upon which the crawler must obey the parsed robots.txt file.

[1] Nissan Hajaj Jesse Alpert. *We knew the web was big...* 2008. URL: `https://googleblog.blogspot.co.uk/2008/07/we-knew-web-was-big.html`.

[2] Free Software Foundation. *GNU Wget.* 2015. URL: `https://www.gnu.org/software/wget/`.

[3] Julian F. Reschke Roy T. Fielding. *Hypertext tranfer protocol, message syntax and routing.* 2014. URL: `https://tools.ietf.org/html/rfc7230`.

[4] Martijn Koster. *Robots Exclusion Standard.* 2007. URL: `http://www.robotstxt.org/orig.html`.

The web crawler must not request too many pages at a time, causing trouble for the website's server.

- A flag must be available to the user to specify a delay before requesting another page from a website's server. If the option is not defined at the command line, the program should default to one request per second. A minimum limit should be applied to the flag to ensure the crawler may not be used for abusive purposes.

The web crawler must provide a flag to limit the depth of the search, and a flag to limit the crawl to a number of domains.

- A flag must be provided to limit the depth of the crawling to ensure that the program does not continue scraping the web almost indefinitely. To this end, another flag may be specified at run time to limit the crawl to either a single domain provided by the user, or the list of domains provided. The web crawler should provide a flag to limit the crawl to the first n domains encountered. The web crawler could provide a flag to limit the crawl to a user specified time period.

Ethical Considerations

Ethical considerations must be made before embarking on this project. Upon crawling some websites, personal data may be encountered, for example when crawling a social media website. To circumvent this issue, any dynamically generated content, links containing the '?' character for example, will be ignored. This is also good web crawling practise, to prevent the crawler from being 'trapped' in a never ending loop. The personal data will not be processed by the web crawler in any way other than to obtain links.

Another ethical concern which must be evaluated is the frequency of downloading the web pages. If the crawler attempts to fetch pages too quickly, the server may consider the crawler to be performing a distributed denial of service attack on the site, and therefore may block connections from the crawler to avoid crashing the server. To correct this issue, the crawler will fetch pages at a sane default of one page every one second per site. This may be overridden at the command line upon launch but it will accept a minimum, to be determined value, to prevent the crawler from being used in an abusive manner.

The final ethical issue which must be assessed is that the website administrator may not wish web crawlers to access certain pages. To comply with this issue, the web crawler will be designed to conform with the robots exclusion standard, and must parse the robots.txt document on every site before attempting to scrape it. The paths defined to be ignored in robots.txt will be ignored.

Work Plan

To ensure that the project is well supervised, I will try and meet my project supervisor once a week. Where circumstances prevent this, I will correspond with my supervisor via email. At some points the work plan may be deliberately vague, as some design decisions may not be finalised until

the project has progressed. Due to the nature of the project I will test as I develop the software, therefore testing will not always be explicitly defined in the plan.

Week 0 - Time up to beginning of Week 1.

- Little research completed.
- Haskell refresher course completed.

Week 1 (Commencing 2016-01-25)

- Initial Plan completed and submitted. - 2016-01-31
- Research methods of parsing HTML in Haskell, with one of the methods investigated being libXML.[5]

Week 2 (Commencing 2016-02-01)

Meeting scheduled for 2016-02-01.

- Complete research into HTML parsing with Haskell.
- Begin research into methods such as HTTP compression to reduce bandwidth use.
- Begin work into robots.txt parser.

Week 3 (Commencing 2016-02-08)

Meeting scheduled for 2016-02-08.

- Complete research into bandwidth use reduction.
- Complete robots.txt parser.
- Begin work on fetching and parsing HTML web pages.

Week 4 (Commencing 2016-02-15)

Meeting scheduled for 2016-02-15.

- Complete work on fetching and parsing HTML web pages.
- Begin work on breadth-first web page fetching.
- Begin research into best cycle prevention algorithm.

[5] Daniel Veillard. *libXML*. URL: `http://www.xmlsoft.org/`.

Week 5 (Commencing 2016-02-22)

Meeting scheduled for 2016-02-22.

- Complete breadth-first web page fetching.
- Complete research into best cycle prevention algorithm.
- Begin work on multi-threading the web crawler.
- Begin implementation of the cycle prevention algorithm found to be optimal during research.

Milestone - The web crawler should now have function as a very basic web crawler.

Week 6 (Commencing 2016-02-29)

Meeting scheduled for 2016-02-29.

- Complete implementation of the aforementioned cycle prevention algorithm.
- Begin work on HTTP compression/other bandwidth saving work.

Week 7 (Commencing 2016-03-07)

Meeting scheduled for 2016-03-07.

- Complete HTTP compression/other bandwidth saving work.
- Complete the multi-threading of the web crawler.
- Begin implementation of the wait flag, constraint flags, and any other flag deemed necessary as the crawler was developed.

Week 8 (Commencing 2016-03-14)

Meeting scheduled for 2016-03-14.

- Complete implementation of command line flags.
- Rent out web space for testing at a large scale.
- Begin testing on a larger scale.

Milestone - The web crawler should now have achieved all of the aims in this plan.

Week 9 (Commencing 2016-04-11)

Meeting scheduled for 2016-04-11.

- Begin final project report.
- Complete testing.

Week 10 (Commencing 2016-04-18)

Meeting scheduled for 2016-04-18.

- Work on report continues.

Week 11 (Commencing 2016-04-25)

Meeting scheduled for 2016-04-25.

- Finish final project report.

Week 12 (Commencing 2016-05-02)

Meeting scheduled for 2016-05-02.

- Submit completed final project report. - 2016-05-06

This work plan is written as a rough guide. There is some leeway in the time allowed due to the one week gap between completion of the report and the submission date, however I am reluctant to stray into that one week period. If any of the work is not completed by the Easter period, then the Easter period will be used as a catch up time period.

Deliverables

Research

Research into optimal cycle prevention method for this problem.

Research into HTTP compression and other bandwidth saving methods.

Final project report.

Code

Haskell based web crawler.

Any extra code that accompanies the respective research deliverable.

Bibliography

Foundation, Free Software. *GNU Wget*. 2015. URL: https://www.gnu.org/software/wget/.

Jesse Alpert, Nissan Hajaj. *We knew the web was big...* 2008. URL: https://googleblog.blogspot.co.uk/2008/07/we-knew-web-was-big.html.

Koster, Martijn. *Robots Exclusion Standard*. 2007. URL: http://www.robotstxt.org/orig.html.

Roy T. Fielding, Julian F. Reschke. *Hypertext tranfer protocol, message syntax and routing*. 2014. URL: https://tools.ietf.org/html/rfc7230.

Veillard, Daniel. *libXML*. URL: http://www.xmlsoft.org/.