In-browser page popularity statistics visualization

Roman Rudenko

Department of Computer Science
University of British Columbia
201-2366 Main Mall
Vancouver BC Canada V6T 1Z4
rudenkor@cs.ubc.ca

Abstract

Most modern web statistics tools format the data in form of standalone reports and graphs. While this approach allows the system to present large amounts of data effectively, it decouples the statistics from the pages themselves, forcing the users to perform URL to page mapping as they analyze a report. We are going to attempt to eliminate this burden from the user by presenting the simple link popularity data inside the analyzed page itself.

1 Introduction

The traditional web statistics packages separate the analyzed website from the obtained statistical data. The users have a choice of many different views, modes, and metrics. However, virtually all such systems share a common flaw – the information is not visible right away to the site maintainer. Instead, in order to gain the general understanding of what destinations are more popular and where do incoming users arrive from, they have to explicitly invoke the statistics package and look through views and reports. We suggest that instead of being treated as separate out-of-band data, web statistics can be delivered to site maintainers in-band, inside the normal web page, and offer a (relatively crude) implementation of this concept.

Obviously, this approach is not capable of, and is not intended to replace the complicated analytics. Instead, it is supposed to provide site maintainers that normally do not refer to statistics, such as artists, editors and writers, with instant basic understanding of relative importance of different site elements. One can draw an analogy with disk usage management – advanced users would prefer to use a separate application to track down and visualize large files and directories, while novice users are more likely to prefer to have basic disk usage data and visualizations inside their normal file browser.

Another important drawback of traditional standalone reporting packages is that during single page analysis, the user is forced to maintain a mapping between text of links inside the page and URLs and page titles that are mentioned inside the report (see illustration below). If the maintainer does not know the site structure by heart, this is going to be a time-consuming chore. If the visualization was presented inside the page, the maintainer would be able to spend this effort on doing their actual job.

2 Related Work

The web statistics company Urchin did offer a similar visualization (fixed-size progress bar style overlays over links). However, it was not clear from their screenshots whether the visualization is performed inside the web page itself, or if an external application is involved. Unfortunately, Urchin was bought out by Google (and became Google Analytics), and virtually all marketing information that was available on their website disappeared. Therefore, I am unable to provide screenshots of this feature in action.

Pages-URL

Total: 208 different pages-url	Viewed	Average size	Entry	Exit
/awstats/cvschangelogbuilder_awstats.html	902	50.68 KB	836	772
/phpwebsite/	897	5.40 KB	345	332
/phpwebgallery/category.php	343	2.54 KB	26	49
/phpwebgallery/picture.php	270	1.86 KB	24	33
/cvschangelogb/cvschangelogbuilder_cvschangelogb.html	234	12.24 KB	163	175
/dolibarr/cvschangelogbuilder_dolibarr.html	115	21.93 KB	82	86
/phpwebgallery/	91	782 Bytes	13	7
/phpwebgallery/admin.php	90	2.00 KB		1
/perso_doc.php	82	5.94 KB	9	13
/dolibarrweb/	57	3.52 KB	4	5
/awstats/	32	1.59 KB	4	3
/files/Doc/Hacking/FAQ/	25	8.88 KB	13	9
/phpwebgallery/identification.php	18	1.10 KB	2	4
/files/Doc/Gestion%20projet/CMMI%20-%20Estimation%20de%20charges...	11	13.84 KB	1	3
/files/Doc/	11	1.07 KB	2	1
/dolibarr/dolibarr-2.0.0-alpha2.tgz	11	1.93 MB	7	9
/files/Doc/Gestion%20projet/Mod%E8le%20-%20R%E9ponse%20appel%20o...	10	179.78 KB	3	2
/files/Doc/Client-Serveur/Bible%20-%20Client-Serveur.sxw	10	44.31 KB	2	

Illustration 1: The traditional statistics package (AWStats) - data is separated from content

3 Description of Solution

The core idea of the solution is to add the statistics server to the normal "client/web server" model, and modify the client to request the popularity data from the statistics server when appropriate. So, the website does not have to be modified at all to make use of our system.

3.1 Implementation details

The statistics are extracted from Apache access logs, processed and served by a Python script running on Apache. We chose Firefox as a client browser (as it offers mature and standards-compliant Javascript support), and modified it by adding the Greasemonkey extension to allow execution of our custom Javascript on required pages. Our script uses the SAJAX library (argument marshaling/unmarshaling wrapper around XMLHttpRequest) to facilitate parameter passing between client and statistics server. The script executes when the page is received from server and rendering is completed. It will extract the links available in the page and pass them to statistics server. The statistics server returns hit counts, which are scaled and visualized as transparent overlays by our script. So, the basic data flow takes the following shape:

- Client receives a page from the web server
- Client Javascript retrieves the links, and obtains corresponding data from statistics server
- Client Javascript renders the overlays

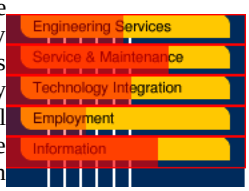
3.2 Data

Originally, we have planned that for every link, we would show number of requests from current page to target page. Unfortunately, this idea was not implemented, both due to lack of time and due to it being not useful on our testing data. Vast majority of visitors arrive via search engines, with second largest group coming "from nowhere" (their requests have no HTTP referrer at all, so they are probably copy/pasting links or using bookmarks). As a result, the identifiable inter-site traffic data was not sufficient to demonstrate a convincing trend, and so we decided to fall back to displaying global hit statistics for every link instead.

3.3 Overlay

The hit statistics are displayed in form of semi-transparent overlays. The overlays are passive and allow no interaction – clicking one would give the same result as clicking the link would. The overlay combines semi-transparent red tinting and progress bar shape. Since we expected that links can differ in popularity by a couple of orders of magnitude, we scale the received hit count logarithmically prior to displaying.

Graphical buttons and tabs often have the same width and are placed side by side, and therefore positions of progress bars that overlay them can be easily compared. Coloured tint is not as useful in this case, as buttons may have coloured or patterned backgrounds which make direct comparisons difficult.



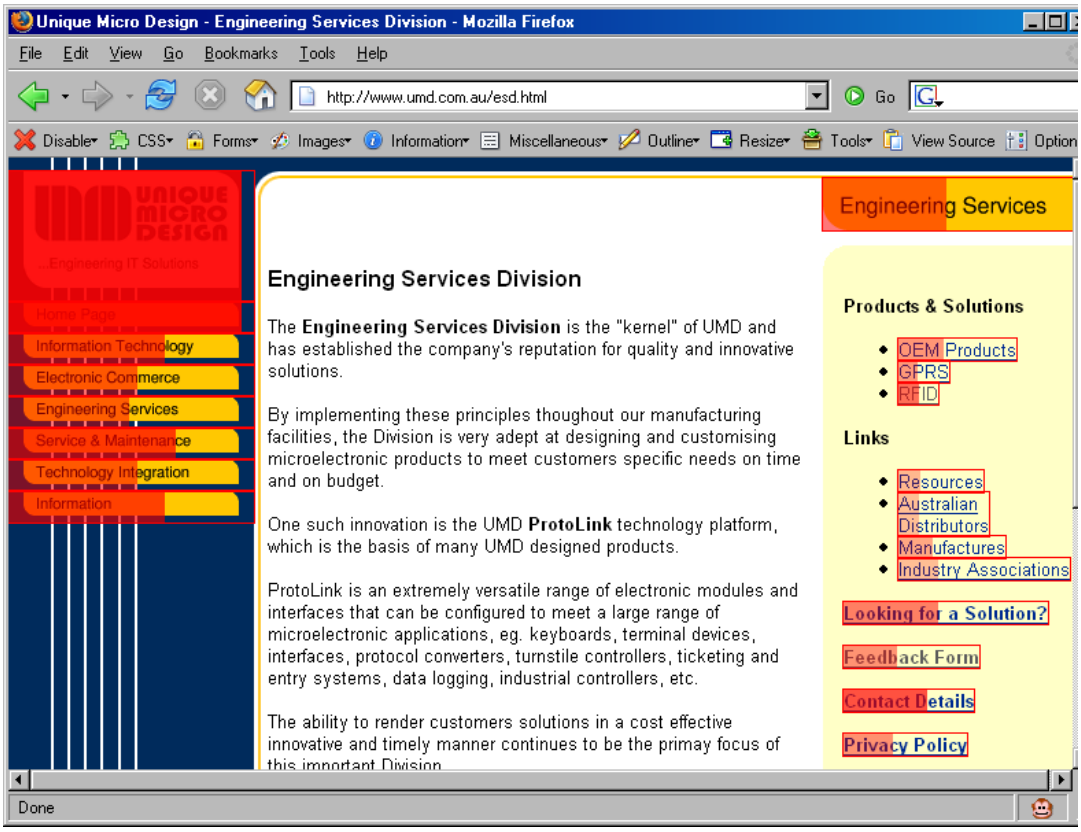


Illustration 2: Our statistics visualization system

Textual links generally have uniform background, and therefore different intensities of coloured tinting would be easy to compare. Meanwhile, their lengths could be different, and so comparisons of progress bar lengths are usually meaningless.

By combining these two features, we get a universal overlay that performs reasonably on most kinds of links. It does has its drawbacks, however. First of all, some users perceived intensity and size of bar fill as two independent variables instead of a single redundantly encoded parameter.

The second problem is that the more frequently visited (and therefore, important) a link is, the harder it is to read it, as overlay becomes more and more intensive. One could attempt to place the tint in background of text instead of over it, but this approach would offer no improvement for image-based links, and it is unclear how one would deal with links that already have a background (CSS is limited to a single background image per element). Radical colour channel manipulations, such as displaying the page in grayscale and highlighting links with red, would also be impossible, as no modern browser offers the ability to perform operations on colour channels. One could hope, however, that future SVG filters and blending modes would provide more compositing options and would allow us to address this shortcoming.

- Product information and pricing

4 Scenarios of Use

This visualization is passive and is expected to provide context for site maintainers. Therefore, the user does not actively use this visualization, but rather refers to it while performing their normal duties, such as content editing and restructuring.

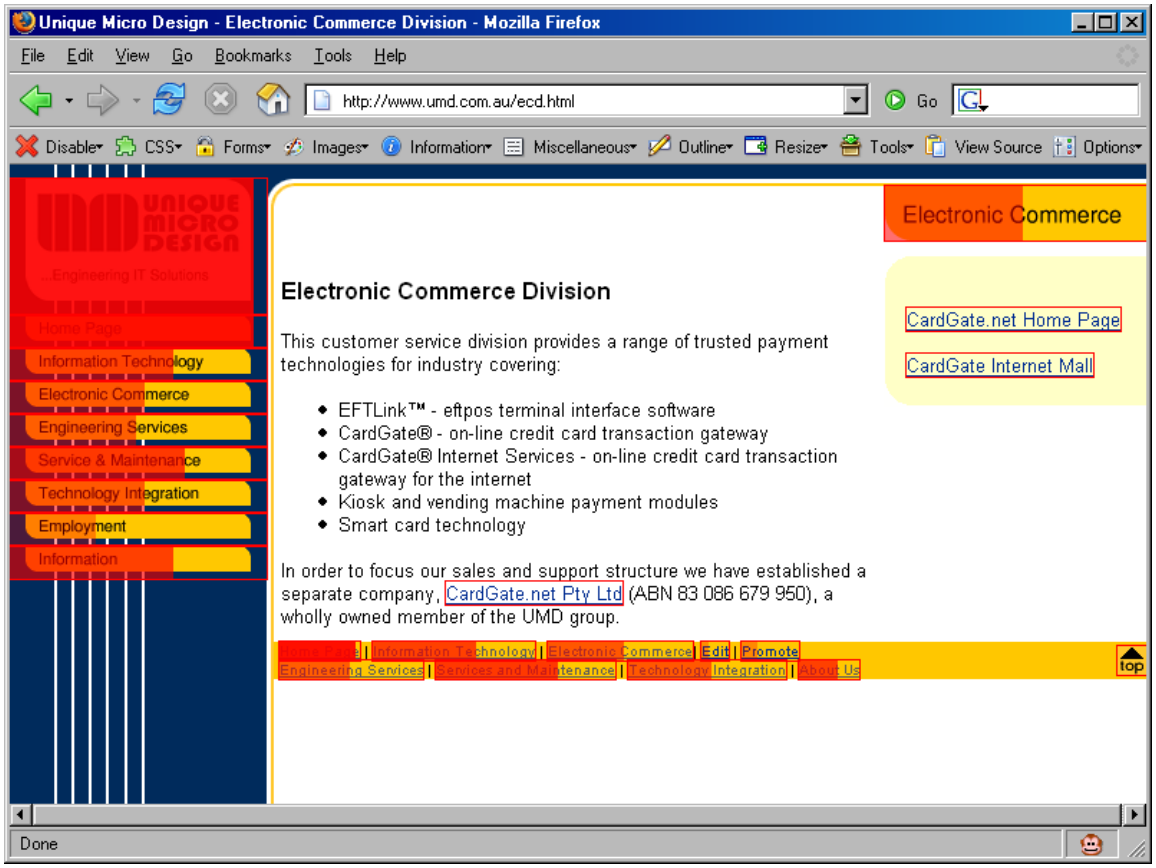
5 Performance/Scalability

The current implementation uses HTTP GET request to obtain the data from statistics server. This means that the total length of URLs in a request is limited by the maximum length of URL that the HTTP server and client can properly handle. Unfortunately, SAJAX library (used for client/statistics server communication) currently does not provide HTTP POST support for its Python bindings. So, maximum total length of processed URLs in a single page is sufficient for a small-scale demonstration, but is too constraining for real-world use.

The performance of the system appears to be limited mostly by DOM object creation and response time of the server component. One can reasonably expect the original page sizes to become an issue before the overlay rendering lag becomes significant

6 Lessons Learned

- Not surprisingly, Javascript programming was proved to be hell once again.
- Currently available browsers are not capable of handling advanced image compositing modes, limiting the choice of available visualization techniques. This issue may be alleviated when browsers would start supporting CSS3 Background module and SVG blending modes.
- First 10% of project takes 90% of expected time. The remaining 90% of project takes another 90% of alloted time.



7 Strengths and Weaknesses

The obvious strength of this project is that does not require the user to cross-reference URLs and page titles to individual links. The barrier to access the data becomes much lower, and therefore the editors and site maintainers that normally do not reference page statistics are able to take them into account. Most web writers do not consult traditional statistics tools when modifying content, but if a visualization would provide them with a quick idea on which links inside a page are popular and which are not, they would be able to perform basic reordering and reprioritization quite easily.

Another strong point of this system is the minimal installation requirement. Client browser requires an addition of one extension and installation of a single script, while the web server requires no changes at all. This would allow the system to work even with legacy content management systems that cannot be modified without significant effort, and

The primary drawback of the system is its incomplete and limited implementation, which prevent it from being useful in real site maintenance. Poor time management resulted in too much effort being spent on backend, and too little on visualization itself.

More fundamental limitations of current approach include possible overlay overlaps if the links themselves overlap, and general unsuitability for dynamic Javascript-driven websites. Adding support for tracking creation, destruction and repositioning of links would require a far more complicated implementation .

Referernce

[1] Google Analytics, http://www.google.com/analytics/

[2] AWStats, http://www.awstats.org/