CS 280 Report

Content Management System Detection in the Yioop Search Engine

By

Charles Bocage

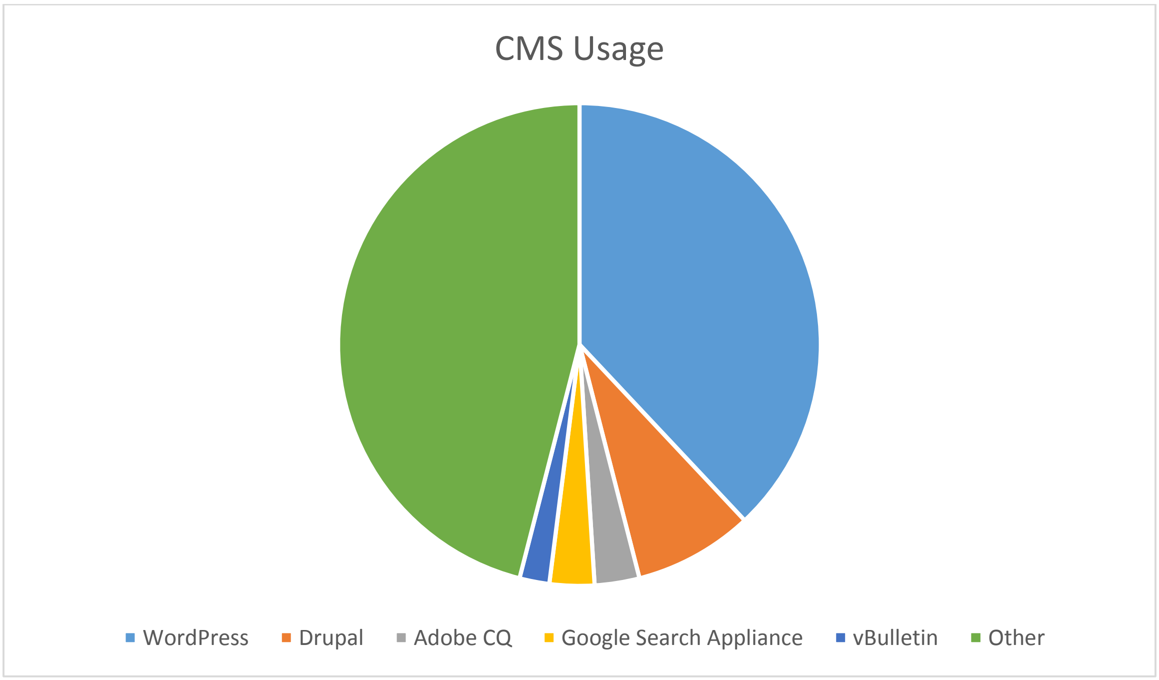
Project Advisor: Dr. Chris Pollett

Department of Computer Science
San José State University
One Washington Square
San José, CA 95112

Table of Contents

Introduction

A CMS is a tool "for building and maintaining web applications for many disciplines" (Mooney & Baenziger, 2007). In other words, a CMS helps organizations stand up web content quickly with virtually no web programming experience. For example, the Herald Sun, a news source from Australia, uses a CMS called WordPress. In addition, "WordPress is used by 25.4% of all the websites" (W3Techs, 2015). WordPress offers its users many plugins, themes, site management functionality etcetera for them to publish content and apply a consistent layout. According to CMS Usage Statistics if I only targeted the top five sites I would capture 54% of the CMSes on the Internet as shown below:



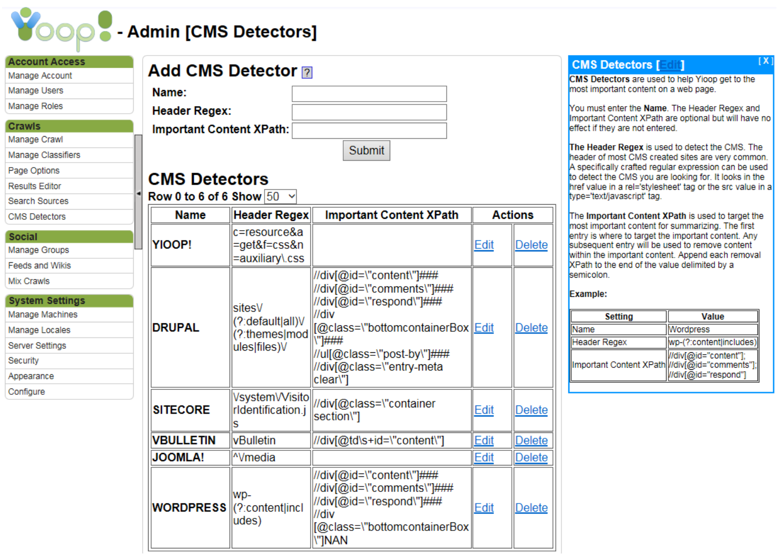
When a search engine crawls a page, it usually extracts the most important parts to store in the index. By detecting which CMS a web page, summarizers can better find the most important content. As the Yioop search engine crawls now, it does not detect which CMS a web page uses. Accurately targeting the important content will allow the Yioop search engine to produce better summaries of the content it has crawled. That is because most CMS developed web pages follow a specific pattern to render the page in a browser. For example, the page content, navigation content, side-bar content etcetera are located in

HTML tags that are decorated with the same names or attributes despite the content within the tags. Furthermore, the important content will also be located in an HTML tag that is common to all of its pages.

This semester I created an abstract CMS detector framework that is flexible enough to detect virtually any CMS based on users' input. The Yioop search engine users can configure the CMS detector settings through a new GUI Activity within the Yioop search engine. The GUI Activity allows users to add/remove CMS detectors without writing any code. I also produced a before and after fetch duration comparison and patches Dr. Pollett can include in his subsequent release of the Yioop search engine.

Deliverable 1: A New GUI Activity

The goal of this deliverable was to create a new GUI Activity in the Yioop search engine to allow users to manage its CMS detector settings. In order to meet this deliverable I needed to create a place for users to enter the CMS detector settings in the Yioop GUI and update the database to store the CMS detector settings. First, I modified the appropriate files to enable an activity called CMS Detectors within the Crawls group. When a user clicks CMS Detectors, the current settings are shown. I kept the CRUD methodology in mind while I was creating the interface. Users will be able to create, read, update and delete CMS detectors from the Yioop search engine. A CMS detector has to have a name and a header regex to be successfully submitted. The important content XPath value can be empty as it is not required. There is help documentation also. If a user clicks on the question mark next to the "Add CMS Detector", a full description of the CMS detectors page is displayed.



- Admin [CMS Detectors]

Add CMS Detector ?

Name:
Header Regex:
Important Content XPath:

CMS Detectors

Row 0 to 6 of 6 Show 50

Name	Header Regex	Important Content XPath	Actions	
YIOOP!	c=resource&a=get&f=css&n=auxiliary\.css		Edit	Delete
DRUPAL	sites\/(?:default|all)\/(?:themes|modules|files)\/	//div[@id=\"content\"]### //div[@id=\"comments\"]### //div[@id=\"respond\"]### //div [@class=\"bottomcontainerBox\"]### //ul[@class=\"post-by\"]### //div[@class=\"entry-meta clear\"]	Edit	Delete
SITECORE	\/system\/Visitorldentification.js	//div[@class=\"container section\"]	Edit	Delete
VBULLETIN	vBulletin	//div[@td\s+id=\"content\"]	Edit	Delete
JOOMLA!	^\/media		Edit	Delete
WORDPRESS	wp-(?:content|includes)	//div[@id=\"content\"]### //div[@id=\"comments\"]### //div[@id=\"respond\"]### //div [@class=\"bottomcontainerBox\"]NAN	Edit	Delete

CMS Detectors [Edit]

CMS Detectors are used to help Yioop get to the most important content on a web page.

You must enter the **Name**. The Header Regex and Important Content XPath are optional but will have no effect if they are not entered.

The Header Regex is used to detect the CMS. The header of most CMS created sites are very common. A specifically crafted regular expression can be used to detect the CMS you are looking for. It looks in the href value in a rel='stylesheet' tag or the src value in a type='text/javascript' tag.

The **Important Content XPath** is used to target the most important content for summarizing. The first entry is where to target the important content. Any subsequent entry will be used to remove content within the important content. Append each removal XPath to the end of the value delimited by a semicolon.

Example:

Setting	Value
Name	Wordpress
Header Regex	wp-(?:content|includes)
Important Content XPath	//div[@id="content"]; //div[@id="comments"]; //div[@id="respond"]

Furthermore, to store the CMS detector data, the Yioop search engine's database has to be modified. I added information about the new activity in the ACTIVITY, TRANSLATION_LOCALE, TRANSLATION and ROLE_ACTIVITY tables. I added the name of the activity to the ACTIVITY table, a translation value to the TRANSLATION table, mapped the ID from the TRANSLATION table to its entry in the TRANSLATION_LOCALE table and mapped a role to its entry in the ROLE_ACTIVITY table. I also created one table called CMS_DETECTORS. The table has 4 columns; TIMESTAMP, NAME, HEADER and IMPORTANT_CONTENT. The GUI activity mimics this table in its display except the timestamp column is not shown. The timestamp column is the primary key and is stored as a hidden value to enable users to edit and delete a CMS detector setting.

Once I had the CMS detector page designed and the database created, I did some stress testing on the changes. I entered long values into each column to see how the CMS Detector table would render the information in the GUI. The table stretched beyond the page bounds causing the data to be unreadable. The data also breached the help content when it was displayed. To solve this problem, I configured the search.css file to make the table cells wrap the text. Another test I performed was to update the data stored in the database. I noticed when I attempted to edit a CMS detector setting the data did not display correctly in the input box. After some debugging, I found that the problem was caused by the value containing quotes. For example, if "1234" (including the quotes) was stored in the database and the user edits that CMS detector, the GUI would try to render value=""1234"". This would cause that value to display an empty value, basically value="", even though you can see it in the source of the page. In order to address this problem, I stored the data in the database HTML encoded. Now the database stores "1234" as &quot;1234&quot; (all other HTML characters are stored as their HTML encoded values as well) and when rendered a browser coverts it to it humanly readable equivalent value="1234".

Deliverable 2: Incorporate CMS Detector into Yioop's Fetcher

The goal of this deliverable was to extend Yioop's fetcher to incorporate the user defined CMS settings. This is where I took the work I did in CS297 and extended it. In CS297, I came up with code for two separate detectors. I had one for WordPress and one for Drupal. Each CMS detector had its own name, unique header regex and important content values. In that model a detector for each CMS would need to be created to detect new CMSes. This would not be very easy to manage, so I had to come up with a better idea.

In deliverable 1, I added the ability to store the CMS values needed into the Yioop search engine's database. With those values in a location that could be queried for, a universal CMS detector could be created. The universal CMS detector queries the CMS_DETECTOR table for the CMSes it is to detect. It loops through each setting until it finds a match. If it does not find a match, it returns UNKNOWN. Within the HtmlProcessor.php file a checkForCMSContent() method was added. Before it summarizes the content, the HtmlProcessor.php code calls the checkForCMSContent() method to see if the content is from a CMS that it recognizes. If the content is from a CMS it is aware of, the important content is return as long as that setting is populated. If the important content value is empty or it is not aware of the CMS, then the entire web page is summarized as is. Now every time the fetcher summarizes a document, it has the best chance of generating a perfect summary.

Deliverable 3: Before and After Adding CMS Detectors Experiment

The goal of this deliverable was to compare summary results before and after adding CMS detectors. I already proved that detecting the CMS improved summary results in my CS299 research, so I decided to compare the time the fetcher takes to summarize the documents it crawls. In order to see a noticeable difference there needed to be a substantial amount of CMS detectors. I wrote some code, when run, would add 239 CMS detectors to the database. I performed the experiment as follows:

1. Ran ManyDetectors,php to add 239 CMS detectors.
2. Ran a crawl for about an hour
3. Captured the fetcher logs
4. Deleted all but twenty CMS Detectors
5. Ran a crawl for about an hour
6. Captured the Fetcher logs
7. Deleted all CMS detectors
8. Ran a crawl for about an hour
9. Captured the fetcher logs

After the experiment was complete I analyzed the results. The results showed that with 239 CMS detectors, document summarization took extremely long. Also with 20 CMS detectors it took four times as long.

	0 CMS Detectors	20 CMS Detectors	239 CMS Detectors
Max	20.328	23.328	89.125
Min	0.016	0	0.203
Median	0.313	3.219	44.094
Mode	1.909	0.031	0.656
Mean	0.859	4.362	34.652

I ran my findings by Dr. Pollett. He was concerned that I may have a bug in my code causing the degraded performance. After a short discussion we determined that the culprit is wither my code loading the head tag inefficiently or an old version of PHP. In my code, for each document to be summarized, I reload the head tag for each CMS to be detected. I was also using PHP 5.63. I modified the universal CMS detector to load the head tag once, upgraded my PHP version and reran on my experiment.

Upgrading my PHP version to 7.05 did not change the time it took the fetcher to summarize the documents. Dr. Pollett was correct about loading the head tag once. The results decreased dramatically. The results showed that with 239 CMS detectors it took two seconds on average. With twenty detectors it took 1 second on average. Since this test went so well, I decided to include six default CMS detectors. I went with WordPress, Drupal, SiteCore, Joomla, vBulletin and Yioop.

	0 CMS Detectors	20 CMS Detectors	239 CMS Detectors
Max	1.469	2.234	4.141
Min	0	0	0.609
Median	0.0313	1.016	2.445
Mode	0	0	3.609
Mean	0.347	1.032	2.361

Deliverable 4: Submit the Patches

The goal of this deliverable was to create a patch that contains my changes and submit it to Mantis BT. Since these items were broken into two categories, GUI and database, I needed to create two patches. I went through the files I modified or created and determined the following files belonged to the following groups:

CMS Activity Patch	Database Patch
PublicHelpPages.php	Config.php
AdminController.php	Createdb.php
Controller.php	UpgradeFunctions.php
CrawlComponent.php	ProfileModel.php
search.css	
UniversalDetector.php	
HtmlProcessor.php	
configure.ini	

CmsModel.php	
CmsDetectorsElement.php	
CmsDetectorTest.php	
ManyDetectorsExperiment.php	
Joomla01.txt	
SiteCore01.txt	
SiteCore02.txt	
Yioop01.txt	
cms_detector_input.txt	
cms_detector_results.txt	
vBulletin01.txt	
vBulletin02.txt	

Once I checked for long lines and cleaned each file I created an issue on the Mantis BT site and uploaded the patches. There was a problem with one of the files in the CMS Activity patch. The help documentation I generated was not compatible with the latest version of the Yioop source code. I cloned the latest version of Yioop, regenerated the help documentation and uploaded the patch again. This time that patch was accepted and my code changes are now live.

Conclusion

In conclusion, adding the CMS detector activity required a lot of work to be done. An activity relies on the database to be modified not just the PHP code that renders the pages. I came up with a sample GUI page and the database table to support it. Performing load testing proved invaluable to exposing a bug in the code. Although the majority of the websites are created by a small number of CMSes, a user may want to split them into various versions, increasing the number of CMS detectors available. If I had not done the load testing to expose that problem, users may have started to complain as they added more and more detectors. The work is now complete for the next person to expand Yioop into the land of being a CMS detector authority.

References

Mooney, S. D., & Baenziger, P. H. (2007, July 31). Extensible open source content management systems and frameworks: a solution for many needs of a bioinformatics group. *Oxford Journals*, *9*(1), 69-74. Retrieved December 12, 2015, from http://bib.oxfordjournals.org/content/9/1/69.full

W3Techs. (2015, November 9). *Usage of content management systems for websites*. Retrieved from World Wide Web Technology Surveys: http://w3techs.com/technologies/overview/content_management/all/