Spring 2011

Smart Search: A Firefox Add-On to Compute a Web Traffic Ranking

Vijaya Pamidi
San Jose State University

Smart Search: A Firefox Add-On to Compute a Web Traffic Ranking

A Writing Project Report

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Vijaya Pamidi

May 2011

Smart Search: A Firefox Add-On to Compute a Web Traffic Ranking

by

Vijaya Pamidi

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Chris Pollett, Department of Computer Science

Dr. Tsau Young Lin, Department of Computer Science

Dr. Robert Chun, Department of Computer Science

ABSTRACT

Smart Search: A Firefox Add-On to Compute a Web Traffic Ranking
by Vijaya Pamidi

Search engines results are typically ordered according to some notion of importance of a web page as well as relevance of the content of a web page to a query. Web page importance is usually calculated based on some graph theoretic properties of the web. Another common technique to measure page importance is to make use of the traffic that goes to a particular web page as measured by a browser toolbar.Currently, there are some traffic ranking tools available like www.alexa.com, www.ranking.com, www.compete.com that give such analytic as to the number of users who visit a web site. Alexa provides the traffic rank for a website based on two factors: The number of users that view a website and the number of pages viewed. The Alexa toolbar is not open-source.

The main goal of our project was to create a Smart Search Firefox add-on for the Yioop search engine, an open source search engine developed by my project advisor, Dr. Chris Pollett. This add-on would provide similar analytic data to the Yioop search engine, but in a transparent and open-source way. With the results received from the Smart Search toolbar extension, the Yioop search engine refines the search results as well as provides user centric-search results. Eventually, users would benefit from these better search results.

ACKNOWLEDGEMENTS

I would like to thank Dr. Chris Pollett for being my advisor, guiding and an excellent support throughout the project. I would like to thank my committee members Dr. Robert Chun and Dr. Tsau Young Lin for accepting to be the committee and the effort and time. Very special thanks for my husband and daughter for their great moral support. I would like to thank Shalini Kodali (SJSU Alumni) for peer reviewing the project report.

Table of Contents

List of Figures

1. Introduction

Search engines use web page ranks to determine the order of search results that should be presented. The ranking of a web site is determined by a search engine based on the ranking algorithm followed by that particular engine. Currently there are some tools available like www.alexa.com, www.ranking.com, www.compete.com and these tools provide analytic data for ranking the web sites based on web traffic and the number of users who visit a web site. Alexa provides the traffic rank for a website based on two factors: The number of users that view a website and the number of pages viewed. Usually the factors that affect ranking of a web site are number of users that visit a web site and the incoming, outgoing link probability. Apart from these two factors we would like to add how user preferred search results affect the web traffic ranking. For this project we created a Firefox add-on to compute a Web Traffic Rank and the personal user search.

Our extension can be installed by a user of Firefox. The Smart Search toolbar captures the link that user clicks. Apart from this link it also captures other user driven actions such as the target link, word that user clicked, timestamp of action performed and language preference. The captured data is sent to Yioop periodically. After receiving the data from toolbar, Yioop writes the data to text files under schedules folder. When Yioop performs a web crawl along the Index data, Schedule data, Robot data folders the Toolbar data folder will also be crawled by the Yioop crawler. All the crawled web pages will be indexed and ranked by Yioop. When a user enters a search query Yioop gives the results. The search results given to the user are based on both the open web crawl and the toolbar data. In this way the user eventually could experience a better search results that are very user specific.

One of the main toolbars that exist so far for traffic ranking a web site that is based on the user interest is Alexa. Alexa toolbar works in such a way that it gives ranking of web site from one to infinity, this means that a website with ranking one is more popular than any other web site and the one with ranking two is next and so on. Once a web site that wants to get an Alexa traffic ranking is crawled by Alexa crawler then it starts giving that web site a traffic rank from one to infinity. As Alexa starts getting user data from those users who have installed or using the Alexa toolbar it takes the analytics coming from those users and starts re ranking the current web site. The analytic data is sent to Alexa from the toolbar on a three month periodic basis. User who would like to know the Alexa traffics ranking for their web site should allow three months of time period to get accurate ranking. One thing to observe here is Alexa ranking for a particular web site is based on the traffic gathered from those users who has Alexa toolbar. There are many other toolbars available which does the similar job. Alexa ranking got more popular because of its early availability and the factors considered for traffic ranking and the Google analytics also does a similar traffic ranking.

The Smart search toolbar also works in a similar way but is particularly developed to give user centric search results for users who search queries through Yioop search engine. Smart search toolbar captures user clicked links and some other information about those links. This data will be sent to Yioop server periodically not on the time basis but on the amount of the data captured. When Yioop server receives the data then Yioop performs a crawl the data from toolbar will also be crawled along with the open web crawl and the web sites that are crawled by Yioop will be given a page rank. Web sites will be ranked based on the continuous synchronized process mentioned earlier .This helps the users to get user centric search results.

The initial part of the project report explains how a Firefox toolbar extension was built, modifications and enhancements done to the toolbar to capture user clicks, and the storage mechanism of the user clicks at the user end. The toolbar was also modified to communicate with Yioop and transfer toolbar data in periodic regular intervals.

In the later part of the report the server side process will be explained. The details of how Yioop receives the toolbar data and processes the data to index and rank the web pages will also

be provided. Yioop gives refined search results to user when user enters a search query. The test results are experimented on the toolbar data with Yioop. The last part gives a conclusion to the research project report.

2. Technologies Used

This section explains the technologies needed for developing the requirements of the project. The main user interface language needed to build any Firefox extension is XUL. One of the main requirements for the project is building a Firefox toolbar extension. In order to make the toolbar functional the JavaScript functions need to be added either to the toolbar or to the toolbar components like menu items. The other major component of this project is Yioop. The components of Yioop server need to be developed in PHP.

2.1 XUL

XUL pronounced as “zool” is an xml user interface language. XUL is used to develop Firefox extensions. XUL runner builds provide the possibility to develop applications on top of Mozilla applications. XUL documents are included in contents, skin, and locale folders. The main XUL overlay component chrome.manifest is placed under contents folder. In skin folder the css styles are included. [1]

The elements that can be added as extension or XUL elements include window, dialogue, page, wizard etc. Elements like button, list box, text box, radio buttons, check boxes, toolbar, and menu can also be added. [1]

Events and scripts include command, script, key etc. Apart from these there are many other elements, and events that can be included with XUL. In this project XUL is used to build the Smartsearch toolbar extension and also to add the JavaScript functionality to the toolbar.

2.2 JavaScript

JavaScript is an object oriented web language that is mainly used to build web application functionalities. In this project most of the functionalities were developed in JavaScript. As the Smartsearch toolbar resides at user end or client side we can say that client side functionalities were developed using JavaScript. The details of how the JavaScript code and integration of the various functionalities work will be explained in the later sections.

2.3 PHP

PHP is a scripting language generally used for "server side web development". PHP runs generally on web server. In the current project the server is Yioop! Search engine. Most of the development for Yioop was done with PHP scripting. The application components that need to be developed to communicate with Yioop server should be coded in PHP. So the server side components development for the current project was done in PHP. [2]

The main goal of the project involves working and understanding knowledge of Yioop search engine. One of the main components for any search engine is its ranking strategy and the algorithm to rank the web pages. In order to gain knowledge of the algorithms we have to know various existing ranking algorithms. For this project we have studied Google's Page-Rank algorithm and modeled the Page-Rank algorithm for a 10X10 matrix where the matrix represents web pages with incoming and outgoing links. The details of the Page-Rank algorithm and implementation are discussed in the next section.

3. Design Challenges

The main components that needed to be developed to achieve the goals of our project were user end toolbar, storing captured data, server side data handling and ranking the crawled toolbar data along with the web crawl. The main challenge involved in the design phase was to handle the data from many downloaded instances of our Smart search toolbar, as they all send data to Yioop server simultaneously. So making the Yioop server scalable to handle these results is challenging. When the data from the toolbar is sent to Yioop and a crawl is performed then indexing these results is another design challenge involved.

4. Study and Model Page-Rank Algorithm

To understand how the Page-Rank algorithm works, we implemented Google's Page-rank algorithm for a 10X10 matrix. The algorithm needs to be developed in JavaScript. This 10X10 matrix is linked in a way to model a group of web pages connected with outgoing and incoming links. The program is checked in such a way that Google matrix should converge on a given matrix. Also the program outputs the Page-rank values for the 10 web pages given. The 10X10 matrix that would model a web with outgoing links represented as 1 is shown in Figure 1 and this matrix is the input matrix for the Page-Rank algorithm modeled.

Matrix showing the links between pages
P0 0 0 1 0 0 1 1 1 1 1
P1 1 0 1 0 1 0 1 0 1 0
P2 0 0 0 1 0 0 0 0 1 0
P3 0 0 0 0 1 0 0 0 0 1
P4 0 1 1 1 0 0 0 0 0 1
P5 0 0 1 0 0 1 0 0 1 1
P6 1 0 1 0 0 1 1 0 0 1
P7 0 0 0 1 0 0 1 0 1 1
P8 0 0 0 0 1 1 0 1 1 0
P9 0 1 1 1 0 1 1 1 1 0

Figure 1: Inputs given for our Page-Rank algorithm

4.1 Page-Rank in brief

Internet is a source of billions of web pages which are accessible to the user. Search engine needs to find a way to give the most relevant web pages to the user by comparing the relevance and importance of all these web pages. A challenging task is to rank these web pages with an algorithm that can incorporate common usage patterns i.e. "a user who visits a web page A is more likely to click the link to page B than the link to page C." [3]

4.2 Algorithm Outline

The Page-Rank algorithm gives the page rank for the 10X10 matrix web pages by applying the power method on the given input matrix. The Google matrix with the properties "Stochastic, Irreducible, and Aperiodic" converges on the given input matrix. Converge is a factor of having a very small deviation in the product of the Google matrix and the input matrix value. In the

Figure 2 how the values of convergence keeps reducing and then finally can observe a small change in the values and then it is said the Google Matrix is converged on the input matrix and the output of the algorithm is the Page-Rank values for the web pages represented in the model matrix. [3]

The algorithm convergence information looks like in Figure2:

CnVRG@:1.0745918259183678
CnVRG@:0.08846305370307411
CnVRG@:0.01588145424478182
CnVRG@:0.0034950271003942757

Figure 2: Algorithm convergence.

Next we show the page rank for all the web pages from P0 to P9 in Figure 2

Page Rank Sorted:

0.4692751458840261
0.4689990979411388
0.4679978916551898
0.4667344180658395
0.465975017562795
0.4653511612795653
0.46487267330571336
0.46405241771945693
0.4611560986667226
0.4544237261337586

Figure 3: Page-Rank output

5. Building a Firefox Extension

The first step to build or work with any extension is to learn the basic steps involved in building an extension. In order to execute this step we built a simple Firefox extension. With this we get to know the directory structure and how the extension should be added to the profiles as extensions. Along with file structure, we also need to understand the way chrome.manifest and install.rdf involve in the extension development.

5.1 Simple Firefox Extension

To build a fully functional Firefox toolbar extension and to understand intricacies involved in the building process we started by building a simple Firefox extension with a Status bar displaying "Hello, World". Initially to learn more about Firefox extension features like "Hello-World" menu pop up with menu items labeled were added initially. An alert message will be displayed for each menu item based on the appropriate click by the user. A “New-Tool” was added to the tool menu pop up to learn about the” insert after” feature. To program any Firefox extension one should have the knowledge of XUL and the XUL documents needed to build extension. [7]

The XUL documents that are essential for the programmers to define an XUL user interface are

Content: “The XUL document and the JavaScript files exist in this folder. The elements of these components define the layout of the user interface for an extension”.

Skin: the CSS and image files are placed under this skin folder. This is responsible for the appearance of the extension.

Locale: The files to make the extension localization available. The language “user-visible strings” are responsible for making software localization easy. [7]

The code block used to build a Firefox extension is given in Figure 4

1.Install.rdf

```
<?xml version="1.0"?>

<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:em="http://www.mozilla.org/2004/em-rdf#">

  <Description about="urn:mozilla:install-manifest">
    <em:id>sampletest@example.com</em:id>
    <em:version>1.0</em:version>
    <em:type>2</em:type>

    <!-- Target Application this extension can install into,
         with minimum and maximum supported versions. -->
    <em:targetApplication>
      <Description>
        <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>
        <em:minVersion>1.5</em:minVersion>
        <em:maxVersion>3.6.*</em:maxVersion>
      </Description>
    </em:targetApplication>

    <!-- Front End MetaData -->
    <em:name>Hello</em:name>
    <em:creator>Tester</em:creator>
    <em:description>hello world extension.</em:description>
    <em:homepageURL>http://www.mozilla.org/</em:homepageURL>

  </Description>
</RDF>
```

Figure 4: install.rdf

2.Chrome manifest

```
content     sample    chrome/content/
overlay chrome://browser/content/browser.xul chrome://sample/content/sample.xul
```

Figure 5: chrome.manifest

3.XUL Code Block

```
<?xml version="1.0"?>


<overlay id="sample"
         xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">


<script type="application/x-javascript"
    src="chrome://sample/content/sample.js" />


 <statusbar id="status-bar">
  <statusbarpanel id="my-panel" label="Hello, World"  />
 </statusbar>

 <menubar id="main-menubar">
    <menu id="hello-menu" label="Hello-World!" insertafter="helpMenu">
       <menupopup>
          <menuitem label="Hello!" oncommand="window.alert('Hello Keerthi');" />
          <menuitem label="World!" oncommand="window.alert('Its Keerthis World');" />
          <menuitem label="Day!" tooltiptext="" oncommand="window.alert('Have A Wonderful Day in This World');" />
          <menuseparator/>
          <menuitem label="ToDay!" tooltiptext="" oncommand="window.alert('Today Is  '+displayDay());" />
          <menuitem label="Time!" tooltiptext="" oncommand="window.alert('Time Now '+displayTime());" />
       </menupopup>
     </menu>
    </menubar>

<menupopup id="goPopup">
         <menuitem label = "World History" oncommand="window.alert('http://www.worldhistory.com/');"/>
        </menupopup>

  <menupopup id="menu_ToolsPopup">
    <menu id="tools-menu" label="New Tool"
        insertafter="javascriptConsole,devToolsSeparator">
      <menupopup>
        <menuitem label = "World Tool" oncommand="window.alert('A Tool? To Fix This World!');"/>
      </menupopup>
    </menu>
</menupopup>
</overlay>
```

Figure 6: sample. XUL the XUL overlay

6. Building a Firefox Toolbar Extension

After getting familiar with building a Firefox extension next step would be to build a Toolbar extension with id “Smart Search”. The function of a toolbar button will be to capture the user clicked link and target link. Along with these two features the timestamp when user clicked on the link and language of the web page would be also captured. XUL code block is shown in Figure 7.

```
<?xml version="1.0"?>

<overlay id="sample"
         xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">


<script type="application/x-javascript"
    src="chrome://sample/content/sample.js" />


  <window id="main-window">

    <toolbox id="navigator-toolbox">
          <toolbar id="tool-toolbar" toolbarname="Smartsearch Toolbar" accesskey="T"
          class="chromeclass-toolbar" context="toolbar-context-menu"
          hidden="false" persist="hidden">

             <toolbarbutton id="toolbar-button" >
             <menulist>
            <menupopup>
              <menuitem id="t3" accesskey="W" label="SmartSearch" value="1" oncommand="linkclick();">
             <menuitem id="t4" accesskey="H" label="sendCaptureTest" value="2" oncommand="sendCaptureTest();" >
           <menupopup>
            <menulist>
         <toolbar>
   <toolbox>

 <window>

<overlay>
```

Figure 7: XUL code for a toolbar

In the XUL code shown in the Figure 7 we can observe that the JavaScript functions are called on the oncommand event. When we click on the menu item Smart search then the function lickclick () is called. At this point the functions are added to the event oncommand to test the working of functionalities. After the toolbar completes the development and completes testing then user the functionality of the toolbar should be automatically loaded when user opens Fire fox browser. With load event the first JavaScript function linkclick is called which in turn gives the flow for the rest of sequence of functions. This is achieved by the document.addEventListner and call the function linkclick () with the event load. The line of code is shown in the Figure 8

```
 <script>
        document.addEventListener("load", function() { linkclick(); }, true);
</script>
```

Figure 8: calling click with load

6.1 Adding Functionality to the Toolbar

The JavaScript function linkclick () is called when user clicks on a link on a web page. This in turn calls the getword () function which captures the user clicked link and target link at initial stage of the project.

In function linkclick () for loop is used to traverse through each link and calls addEventListner function which has three parameters ("event", function, true) that could be passed. In the present case the three parameters are ("click", getword, true). The "getword" is another function where the user clicks an URL or a link and target links are captured. The command window.content.location.href was used to capture user clicked link or URL and event.target.href is used to capture target links. The results would be stored in an array for future use. The JavaScript showing the linkclick and getword function is at initial stage. The JavaScript functions code for linkclick and get word is shown in Figure 9.

```
function getword(event){

  var content=new Array();
  content[0]= window.content.location.href;
 content[1]=event.target.href;

  alert("lick clicked"+ content[0]);
  alert("target link"+ content[1] );


   /* var content=window.content.location.href;
    alert(content);
    var content1= event.target.href;
    alert(content1);*/

          var intervalID = window.setInterval(linkclick, 1000);


           Write("http://localhost/yioop/ajax.php", "file=ajax-post-text.txt&content=" + content );

     }



function linkclick()
{
 // alert("In function2");

  var len = content.document.getElementsByTagName("a");
   //alert("a elements in this  "+len.length);

     for (var i=0; i<len.length;i++)
     {
        len[i].addEventListener("click", getword,true) //invoke function
      }
}
```

Figure 9: linkclick and getword functions

6.2 Adding Styles to the toolbar

Making the toolbar button or an extension look good is one of the tasks of building a viable toolbar extension. For this we added the Yioop search engine icon to the Smart search toolbar button. In order to add the icon to the toolbar button we should include the skin folder under our extension folder. The skin folder should be placed at the same hierarchy as content sub folder under the chrome folder. The skin folder should hold the Yioop search engine icon which in this case is named as favicon.jpg and this is downloaded from the Yioop. The skin folder should also contain the css file that is required to add the styles for icon. For this toolbar the css file is named as test.css. The XUL file should contain the path for the test.css file. After adding the Yioop icon the toolbar button looks like in the Figure 10 shown below.

Figure 10: Toolbar button with Yioop icon

6.3 Yioop! Code study

The goal of the project is to provide user specific search results with the Firefox toolbar extension. In order to do that we need to merge the user captured toolbar data with the Yioop index archive. To understand the work flow of Yioop search engine it is mandatory to study the Yioop code. The goal of code study is to understand the flow of functional events taking place in Yioop. This starts at the point where user enters search query in the Yioop search bar and ends at the point where Yioop returns the results back to the user. To understand the working of Yioop code in a better way, a small task like changing the color of a part of the search results was done.

The matching word in the results for the search query was changed to color green. For example if user entered the word “sjsu” in search text field, each “sjsu” word in the results was changed to color green. These changes are shown in Figure 11 and Figure 12.

The Yioop results before change.

Figure 11: Yioop before code change

The Yioop results after the color change.

Figure 12: Yioop after code change

Yioop is an open source search engine and it was developed in PHP. The main goal and advantage of Yioop is that user can have control over the crawls done with Yioop if Yioop is already installed locally on user’s system. This helps users to get results from a known set of data rather from a massive crawl data by the popular search engines. Even from the Yioops website user can select the crawls from which he/she would like to get results from and can also select a mix of crawls as well. To get the advantages of Yioop search engine and the search results from

Yioop one should have knowledge or should learn about the crawling and index mechanisms done by search engines. [4]

The main entry point for Yioop is index.php which is the search engine main web page. This takes the query from user which is entered in the search query text box. The main folder that contains programs for the crawl is bin folder. This contains fetcher.php and queue_server.php. These two are needed to run the crawl after the initial installation and also do other crawls in future. Creation of WORK_DIRECTORY can be done when user signs-in as admin with the given user name and pass word. The user interface provides the availability to configure a work directory, manage crawls, manage roles, mix crawls, manage locales, manage users and manage accounts. The WORK_DIRECTORY set by user contains all the folders and the crawl sites and information in these folders. The Yioop configs folder contains all the necessary programs to carry out these tasks. Once the configuration is set and when user executes a crawl the controllers that are needed to perform the crawl and other activities exist in controllers folder in Yioop. Most of the requests made to Yioop comes to the entry point index.php and the query string after the "?" in the url informs the controller that is responsible for the specific request. The query request made contains a variable c= which tell Yioop which controller should be used to process the request. The arg= variable in the query request string tell which data should be retrieved to get the appropriate results and the models needed for the data and finally the views with which the results should be given back to users.[4]

The other folders like css, data, lib, locale, models, tests and views the basic styles used for Yioop, WORK_DIRECTORY information, scripts contain JavaScript files used by Yioop etc.

Schedules folder under work directory contains the folders with the time stamp in the folder name. The three main sub folders that schedule folder contain are index data folder, robot data folder and schedule data folder. All these folders contain the crawl data gained by the fetcher when it runs a crawl. This data will be processes by queue_server at later point during crawl. The information that comes from robot.txt files is stored in robot data folder. The schedules data folder contains the sites that are found during the web crawl and theses sites will

be later crawled by the crawler. As the sites are crawled these are added to the mini inverted index first and then later to the global index. The indexes are stored in Index data folder. [4]

Once the crawl is completed with the folders in the schedule folder the cache folder contains the entire cache of the web pages that were crawled. The cache folder contains Quebundle folder, Archive folder and Index data folder. The index data folder contains summaries and dictionaries folder inside. To serve search results Yioop relies on all of these folders. The queue_server.php is responsible for the Quebundle and Index data folders. Quebundleis responsible for the priority information for the queue to be processed during crawls. The cache of web pages crawled by the crawler are stored in Archivebundle folder. [4]

6.4 Making a Toolbar Extension Communicate with Yioop

After capturing the user clicked links the next step would be to send the captured data to Yioop server. This brings the need for communicating with Yioop into picture. At this point, data is sent to Yioop instantly whenever user data is captured without storing it at user end.

As part of making communication with Yioop, a POST request is made to Yioop from toolbar button. All the links sent to Yioop are saved in a text file and a link is added to access this text file. To achieve this JavaScript was used at user end and PHP was used for POST at the server end which in this case was Yioop. As this was the development phase of the project, the Yioop code was located at the root folder at local host and the server side programming was done in the local host for deployment. The JavaScript function “Write” was coded to achieve this task. The function has got two arguments passed to it. One is URL and the other is content. This function was invoked at the end of getword function.

The function “Write” has “createXHR” function called in it. The “createXHR” was to establish an http request, if the “readyState ==4” is yes then a connection will be opened to the given URL and makes a POST request. Then it will POST the content to the URL, which incase here it is http://localhost/yioop/ajax.php.

At server side the ajax.php is responsible for the POST request and to capture the content sent from the toolbar button. Once the content is received, the PHP program creates a

text file with the given name in the program and writes the content into the file. Once writing is done it closes the text file. Writing to the text was done in a+ mode. It is an append mode used for both writing and reading the content to the file. The a+ mode was used keeping the future project development in view.

Accessing this file is done from Yioop index page instead of accessing it with typing the location of the file in the browser. A link is created with name “Activity” in the index page of the Yioop code. When clicked on the link it navigates to the page where links were saved.

The code snippet is shown in Figure 13:

JavaScript code where “Write” function is invoked in “getword” function

```
function getword(event){

  var content=new Array();
  content[0]= window.content.location.href;
 content[1]=event.target.href;

  alert("lick clicked"+ content[0]);
  alert("target link"+ content[1] );


   /* var content=window.content.location.href;
    alert(content);
    var content1= event.target.href;
    alert(content1);*/

        var intervalID = window.setInterval(linkclick, 1000);


         Write("http://localhost/yioop/ajax.php", "file=ajax-post-text.txt&content=" + content );

    }
```

Figure 13: Communicate with Yioop

JavaScript code that shows how the “createXHR ()” and “Write ()” functions were implemented is given in the code shown in Figure 14.

```
function createXHR()
{
    var request = false;
        try {
            request = new ActiveXObject('Msxml2.XMLHTTP');
        }
        catch (err2) {
            try {
                request = new ActiveXObject('Microsoft.XMLHTTP');
            }
            catch (err3) {
                try {
                        request = new XMLHttpRequest();
                }
                catch (err1)
                {
                        request = false;
                }
            }
        }
    return request;
}

function Write(url, content)   // url is the script and data is a string of parameters
       {
               var xhr = createXHR();

               xhr.onreadystatechange=function()
               {
                       if(xhr.readyState == 4)
                       {
                               // nothing for now
                               alert("sent " + url + " " + content);
                       }
               };
               xhr.open("POST", url, true);
               xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
               xhr.send(content);
               //alert("after send content" + content);

       }
```

Figure 14: Write and Ajax calls

Ajax.php code that shows how the POST was handled and how the content was written into the text file is shown in Figure 15.

```
<?php

$posted = &$_POST;

$fname=$posted["file"];


if(strcmp($fname, "ajax-post-text.txt") != 0)
    die("You are not authorized to change this file.");

$value = $posted["content"];


$nfile = fopen($fname, "a+");

if($nfile != false)

{

    fwrite($nfile, $value );
    fclose($nfile);

}

?>
```

Figure 15: Write at Yioop with POST

7. Capture and Store User Search History

User data that is captured by toolbar needs to be stored at user end. After storing a certain amount of data based in the conditional check the data will be sent to Yioop. We have chosen SQLite database to store the user end captured data.

7.1 SQLite in brief

SQLite is a light weight database and stores the data in the Firefox profiles folder. As the extension that is being built is a Firefox extension using SQLite database the storage of the user captured data in a SQLite database table would be appropriate.

Most of the commands that work for SQL work in SQLite as well. The knowledge of how to access the SQLite database from command line is required. To access SQLite from command line we should go the default directory in the profiles folder. Profiles folder exists in Mozilla, Firefox. After going to the defaults directory we give the command > sqlite3 example_databse.sqlite, here example_database is the name given to the database we want to store the data tables. In this database we can create, insert and retrieve tables. The database that we use to store the user end data is user_searchcapture.sqlite and the table is user_capture.

7.2 Storage API

To be able to work with SQLite database through the toolbar we need an API. Storage is an API for SQLite database. This can be used with only trusted components like Firefox extensions and components. [5]

7.2.1 Steps in Storage API:

Opening a connection with user_seaarchcapture.sqlite is shown in Figure 16

```
var file = Components.classes["@mozilla.org/file/directory_service;1"]
                             .getService(Components.interfaces.nsIProperties)
                             .get("ProfD", Components.interfaces.nsIFile);
file.append("user_searchcapture.sqlite");

var storageService = Components.classes["@mozilla.org/storage/service;1"]
                               .getService(Components.interfaces.mozIStorageService);
var mDBConn = storageService.openDatabase(file); // Will also create the file if it does not exist
```

Figure 16: Code for Opening an API connection

The code snippet above creates user_searchcapture.sqlite database in profile directory.

Creating Statements and Binding Parameters

After database is created the need arises for the creation of the table to store the data. The sequence of statements for creation and insertion of data into the table are done with CREATE; INSERT commands. The commands are shown in Figure 17

```
mDBConn.executeSimpleSQL("CREATE TABLE IF NOT EXISTS search_capture (word TEXT, searchurl TEXT, searchurl1 TEXT,
                                                                     timestamp TEXT, language TEXT)");

var stmt = mDBConn.createStatement("INSERT INTO search_capture (word,searchurl,searchurl1,timestamp,language)
                                                                  VALUES(:word1,:url1,:url2,:time1,:lang1)");

var params = stmt.newBindingParamsArray();

stmt.params.word1 = event.target.innerHTML;
stmt.params.url1 = window.content.location.href;
stmt.params.url2 = event.target.href;
stmt.params.time1 = new Date();
stmt.params.lang1 = "en-US";
stmt.executeAsync();

void commitTransaction();
```

Figure 17: create and insert in Storage SPI

In the above code block above search_capture table is created with create statement. Insert statement inserts the values into the table. In storage API the values to be stored in the table need to bind with parameters. stmt.BibindingParamsArray(); is used to bind the parameters with the values. This is to increase the efficiency of the working of statements such as create and insert.

At this point we included all the data values that we would like to capture when user clicks on a link. We captured the word that user clicked on with “event.target.innerHTML” , the link that user clicked and the target links are captured with window.content.loacation.href” and “event.target.href” respectively. Apart from these we are capturing the time stamp that user clicked on the link with “newDate()” function and the language of the web page that user clicked on is captured with content.document.getElementsByTagName("html")[0].getAttribute("lang"). All this data is captured at this point with a view point of Yioop search engine’s indexing and ranking strategies. After this “stmt.executeAync();” executes all the statements and this results in creating table, inserting values into table.

8. Send User Search Capture To Yioop!

After capturing the required data with user click, next step is to send the data stored in SQLite database table to Yioop periodically. Initially at the development stage the data amount we would like to set the condition and send to Yioop is 10 rows in the search_capture table. This means Once 10 rows are inserted into the table the data will be sent to Yioop. Once the development is completed this will be set to 50 rows. The component that sends the data from toolbar to Yioop is sendCapture function.

The code block is shown in Figure 18

```
var file = Components.classes["@mozilla.org/file/directory_service;1"]
                     .getService(Components.interfaces.nsIProperties)
                     .get("ProfD", Components.interfaces.nsIFile);
file.append("user_searchcapture.sqlite");

var storageService = Components.classes["@mozilla.org/storage/service;1"]
                        .getService(Components.interfaces.mozIStorageService);
var mDBConn = storageService.openDatabase(file); // Will also create the file if it does not exist

var colnew = new Array();
var statement = mDBConn.createStatement("SELECT * FROM search_capture");

statement.executeAsync({
        handleResult: function(aResultSet) {
                var i = 0;
                let row = aResultSet.getNextRow();

                for (var row = aResultSet.getNextRow(); row; row = aResultSet.getNextRow()){

                        colnew[i] = row.getResultByName("word") + "|:|" +row.getResultByName("searchurl") + "|:|" +
                              row.getResultByName("searchurl1")+ "|:|" + row.getResultByName("timestamp") +  "|:|" +
                              row.getResultByName("language") + "\n";
                        ++i;
                }

                if(colnew.length >= 10)
                {
                        uploadAsyc(yioopurl, colnew);
                }
        },
        handleError: function(aError) {
                alert("Error: " + aError.message);
        },

        handleCompletion: function(aReason) {
                if (aReason != Components.interfaces.mozIStorageStatementCallback.REASON_FINISHED)
                alert("Query canceled or aborted!");
        }
});

void commitTransaction();
```

Figure 18: Component to read data from table and send to Yioop.

8.1 Component that communicates to Yioop! Periodically

To retrieve the rows from table the initial steps of opening the connection with SQLite database has to be done. Then the statement (“SELECT * FROM search_capture”) would be executed. As this statement needs to be executed asynchronous and should return the result set,

we need to loop through the rows and store the rows in an array. The colnew array stores the rows retrieved from table while looping through the result set. After retrieving the rows from table the condition clonew.length is checked, if the condition returns true check the function "uploadAsync();" is called. This function is responsible to send the user capture data to Yioop. Data is sent to Yioop with a POST request. When sending data to Yioop we have to make a controller that receives the data send by toolbar. This controller is toolbar_controller.php.

The flow of functional events at Yioop starts with idex.php and this calls the responsible controllers base on the request made. So this point should be taken into consideration while sending data to Yioop.

The code block for uploadAsync() is shown in Figure 19

```
function uploadAsyc(url, record){ // url is the script and data is a string of parameters

params = "c=toolbar&a=toolbarTraffic&b=" + record;

        var xhr = createXHR();
        xhr.onreadystatechange=function(){
                if(xhr.readyState == 4)
                        {

                                alert(xhr.responseText);
                                deleteRows();

                        }
        };
        xhr.open("POST", url, true);
        xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
        xhr.send(params);

}
```

Figure 19: UploadAsync function

The POST request made by the uploadAsync() is an Ajax call. The line of code – params = "c=toolbar&a=toolbarTraffic&b=" + record; makes a request POST to Yioop's index.php which recognizes "c=toolbar" as a legitimate user request as toolbar is added in the available controllers.

This takes the flow of function to toolbar_controller.php which again checks for "a=toolbarTraffic" to process the request received. Then the data value at "b=" + record" will be written into a text file by toolbar_controller. Once the file is closed after writing record, a response text is sent back to the uploadAsync function. After the response text is received the deleteRows() function will be called which is responsible to delete all the rows from table

search_capture. And again the process of capturing data, storing in the database, retrieving from table, checking the periodic condition, upload to Yioop and deleting rows from table are all done sequentially after a successful upload.

The code block for deleteRows() is shown in Figure 20

```
function deleteRows()
{
    var file = Components.classes["@mozilla.org/file/directory_service;1"]
                         .getService(Components.interfaces.nsIProperties)
                         .get("ProfD", Components.interfaces.nsIFile);
        file.append("user_searchcapture.sqlite");

        var storageService = Components.classes["@mozilla.org/storage/service;1"]
                             .getService(Components.interfaces.mozIStorageService);
        var mDBConn = storageService.openDatabase(file); // Will also create the file if it does not exist


            var statement = mDBConn.createStatement("DELETE FROM search_capture");
            statement.executeAsync();
}
```

Figure 20: Code block for deleteRows function

8.2 Yioop's Toolbar controller

The component at Yioop end that is responsible for receiving the POST request made by toolbar Firefox extension is toolbar_controller.php. The main functionality of this component is to process the POST request and writes the data into text files with the given directory structure and the filename.

The directory that we would like to place the toolbar data is CRAWL_DIR."/schedules/"."ToolbarData"

The ip address of the computer from which toolbar data is noted with $_SERVER['REMOTE_ADDR'].

Under Toolbardata folder a day folder with the day it received the data as name of the folder is created. This is done as $day = floor($time/86400). Once this is done under the day folder a text file with the name

$data_hash = crawlHash($data_string);

$fname= $dir."/At".$time."From".$address."WithHash$data_hash.txt";

After this step we open the file in a+ mode, write the toolbar data in file and close. Once we close the file a response text “OK TEST” will be sent back to toolbar extension.

Writing the flies with filename in this specific format is to make the synchronization with the other folders under schedules folder. The folders that exist under schedules folder will be crawled by the Yioop crawl when we run a crawl from Yioop.

The code block for toolbarTraffic function in toolbar_controller.php is shown in Figure 21

```
function toolbarTraffic(&$data_string)
    {
        $toolbar_data = $_POST["b"];
        $time = time();

        $dir = CRAWL_DIR."/schedules/"."ToolbarData";

        $address = str_replace(".", "-", $_SERVER['REMOTE_ADDR']);
        $address = str_replace(":", "_", $address);

        $day = floor($time/86400);

        if(!file_exists($dir)) {
            mkdir($dir);
            chmod($dir, 0777);
        }

        $dir .= "/$day";
        if(!file_exists($dir)) {
            mkdir($dir);
            chmod($dir, 0777);
        }
        $data_hash = crawlHash($data_string);

        $fname= $dir."/At".$time."From".$address."WithHash$data_hash.txt";

        $fh = fopen($fname, "a+");
        fwrite($fh, $toolbar_data);
        fclose($fh);
        echo "OK TEST";

    }
}
```

Figure 21: toolbar_controller.php

9. Yioop Component Ranking and Refining

Once data is in the text files with the required name format in the schedules directory and under Toolbar data folder this means that toolbar data is ready to be crawled by the fetcher.php and processed by queue_server.php. When a crawl is performed the data exists in the schedules directory and its subfolders including the toolbar data folder get crawled. This is achieved by

adding the necessary functions to the queue_server.php. These functions are to do the crawl of the data exists in the schedule folder under WORK_DIRECTORY and in scheduled folder find the "Toolbardata " folder and the text file exist in the toolbar data folder. The function that is responsible for doing this task in the queue_server.php is shown in the Figure 22

```
    /**
         * Sets up the directory to look for a file of unprocessed
         * index archive data from toolbar then calls the function
         * processDataFile to process the oldest file found
 */
function processToolbarData()
        {   echo " In the function processToolbarData";
            crawlLog("Checking for toolbar data files to process...");

            $index_dir =  CRAWL_DIR."/schedules/".
                "ToolbarData";
            $this->processDataFile($index_dir, "processToolbarDataInvertedIndex");
            crawlLog("done.");
            echo " End of the function processToolbarData";
        }
    /**
             * Builds the MiniInvertedIndex for the files recived from
             * extension toolbar then adds it to the INVERTED INDEX.
 */
```

Figure 22: processToolbardata function

In the function processToolbarData() the $index_dir tells the queue server too look for the assigned path and this path is given as an argument to the function processDataFile($index_dir, "processToolbardataInvertedIndex"). The function processDataFile is a generic function to find the sub folders under schedules directory and call the appropriate call back method. This function returns the $file which to the processToolbardataInvertedIndex function which process the data in the file and builds a ToolbarInvertedIndex. The code for function processDataFile is shown in Figure 23

```
/**
 * Generic function used to process Data, Index, and Robot info schedules
 * Finds the first file in the the direcotry of schedules of the given
 * type, and calls the appropriate callback method for that type.
 *
 * @param string $base_dir directory for of schedules
 * @param string $callback_method what method should be called to handle
 *      a schedule
 */
function processDataFile($base_dir, $callback_method)
{
    $dirs = glob($base_dir.'/*', GLOB_ONLYDIR);

    foreach($dirs as $dir) {
        $files = glob($dir.'/*.txt');
        if(isset($old_dir)) {
            crawlLog("Deleting $old_dir\n");
            $this->db->unlinkRecursive($old_dir);
            /* The idea is that only go through outer loop more than once
               if earlier data directory empty.
               Note: older directories should only have data dirs or
               deleting like this might cause problems!
             */
        }
        foreach($files as $file) {
            $path_parts = pathinfo($file);
            $base_name = $path_parts['basename'];
            $len_name =  strlen(self::data_base_name);
            $file_root_name = substr($base_name, 0, $len_name);

            if(strcmp($file_root_name, self::data_base_name) == 0) {
                $this->$callback_method($file);
                return;
            }
        }
        $old_dir = $dir;
    }

}
```

Figure 23: processDataFile function

Once the file to be crawled is returned from the processDataFile function to the processToolbardataInvertedIndex() function first reading the data from the file which contains the user captured data from the toolbar exists. While reading the file with file_get_contents, the first step is to read the file by the row delimiter which is a “,” in this case, with the explode function. This returns the lines of the file into the $row variable which we take it as rows for better reading purpose. The next step s to read these rows data with the other delimiter and in this case "|:|" with the explode function again. This gives the entire user captured data into array which in this case is $tok. The values from the $tok array are assigned to the $site[self::LINKS][$tok[2]]= $tok[0] which assigns the target link captured by user clicks, then $site[self::TIMESTAMP]= $tok[3] is assigned which is the timestamp of the user click the last value the language is assigned to the $site[self::ENCODING]= $tok[4] variable. The functional code to achieve this task if the function is shown in the Figure 24

```
$rowdelimiter = ",";
$delimiter = "|:|";
$filecontent = file_get_contents($file);

        $rows = explode($rowdelimiter, $filecontent);
        foreach ($rows as $newrow) {
                $tok = explode($delimiter, $newrow);

                $site[self::LINKS][$tok[2]]= $tok[0];
                $site[self::TIMESTAMP]= $tok[3];
                $site[self::ENCODING]= $tok[4];

            }
```

Figure 24: reading the file

Once the reading process from the file is done and the values captured are assigned to the $site variable set the next step is to build a toolbar_shard from these values. This process is to read the information about the urls in the self::LINKS. After reading the information the values are added to $seen_sites array as summaries. Reading the values of $sites and adding it to the $summaries then storing it in the $seen_sites array is shown in the Figure 25

```
$toolbar_shard = new IndexShard("toolbar_shard");
        $seen_sites = array();
        foreach($site[self::LINKS] as $url => $link_text) {
            if(strlen($url) > 0) {
                $summary = array();

                $had_links = true;

                $link_text = strip_tags($link_text);
                $link_id =
                    "url|".$url."|text|$link_text|ref|".$site[self::URL];

                $link_keys = crawlHash($url, true) .
                      crawlHash($link_id, true) .
                      crawlHash("info:".$url, "true");

                $summary[self::HASH_URL] =  $link_keys;
                $summary[self::URL] =  $link_id;
                $summary[self::TITLE] = $url;
                   // stripping html to be on the safe side
                $summary[self::DESCRIPTION] =  $link_text;
                $summary[self::TIMESTAMP] =  $site[self::TIMESTAMP];
                $summary[self::ENCODING] = $site[self::ENCODING];
                $summary[self::HASH] =  $link_id;
                $summary[self::TYPE] = "link";
                $summary[self::HTTP_CODE] = "link";
                $seen_sites[] = $summary;
```

Figure 25: adding to summary

After the summary is in the seen_sites array then the document words are extracted from the links and are added to the function addDocumentWords().This step is shown with the code in the Figure 26

```
$link_word_counts =
    PhraseParser::extractPhrasesAndCount($link_text,
        MAX_PHRASE_LEN, $lang);

$toolbar_shard->addDocumentWords($link_keys,
    self::NEEDS_OFFSET_FLAG,
    $link_word_counts, array());
```

Figure 26: adding documentwords

With this the data from the Toolbardata files are added to the summaries folder in the Index data folder in the cache folder of the work directory. The next is to create generations where the words from the links are added to the dictionaries folder in Index data folder under cache folder under work directory. To achieve this step the function initGenerationToAdd is called and the toolbar_shard is given an argument to this function. Once adding the words to generation then the function changeDocumentOffsets is called and summary_offsets is given as an argument to this function and the result is stored in toolbar_shard. To achieve this process the how the functions are called and how summary_offsets are read is shown in Figure 27

```
$visited_urls_count = 0;
$generation =
$this->index_archive->initGenerationToAdd($toolbar_shard);

$summary_offsets = array();
        if(isset($seen_sites)) {
                $this->index_archive->addPages(
                $generation, self::SUMMARY_OFFSET, $seen_sites,
                $visited_urls_count);
                foreach($seen_sites as $site) {
                        $hash = $site[self::HASH_URL];
                        $summary_offsets[$hash] =
                        $site[self::SUMMARY_OFFSET];
                        }
                }
    $toolbar_shard->changeDocumentOffsets($summary_offsets);
```

Figure 27: adding to summary offsets

After the reading step and adding to the toolbar shard next step is to add the toolbar_shard to the index_archive and then add the current shard dictionary once save and add is done then the merge All tires tales place which add the words to the merges to the dictionary. For achieving these steps a set of functions need to be called this set is

```
$toolbar_shard->changeDocumentOffsets($summary_offsets);
$this->index_archive->addIndexData($toolbar_shard);
$this->index_dirty = true;
unlink ($file);
```

9.1 User gets refined results for their search queries

When a crawl is done with all the steps given in the previous section the toolbar data is ready for the user to return the results. The results are retrieved from the Index data folder under the cache folder which located in the work directory set by the user. With the functions added to the queue_server.php data will be stored in the summaries and dictionaries folders in the Index data folder in the cache. After finishing the crawl and when that particular crawl is set as the index then results generated for the user search queries are returned from that particular Index data folder. Thus user can perform various crawls and can get the results from that particular crawl.

10. Testing the Toolbar

The basic testing necessary to test the main functionality of our project is to check if the Yioop is returning the results based on the toolbar data. Return of results based on the toolbar data is dependent on synchronization of many factors such as data capture by tool bar, data transfer from tool bar to Yioop, data crawl by Yioop crawler, and data indexing.Process starts when the Smart search toolbar captures the data properly. Then the data should be received by the Yioop and Yioop crawler should be able to crawl the toolbar data and index the data to return the results. To make the testing process more effective we included only one url was included in the seed sites list to be crawled at the manage crawl user interface at Yioop server and that site is www.ucanbuyart.com this url should be returned only for the search query “art” and the rest of

the results should be from the toolbar data from Smart search toolbar. All the tests are run at local host during the development stage which is http://localhost/yioop/.

Test 1

This test is conducted to check the basic functionality of the project requirement which is to verify that toolbar data captured based on user clicks will be returned in the results. Results can be returned only after the Yioop crawl of the toolbar data and indexing of the pages. In order to do the test execution, the data when user browsed the ajpm-gold web site was captured. The result was tested for the test query "Silver bullion". The test results show that the requirements were met. In the Figure 28 we can observe the results returned by the localhost Yioop.

Figure 28: Test1 returning results

Test 2

Now we can test one more step further and see how the toolbar data helps to get better results. For this test the scenario we choose to test if the crawl performed with toolbar data can

return the results for those search queries where we are not yielding any search results at www.yioop.com. For this first we entered the search query "Padmini Paladugu" at www.yioop.com and it returned 0 results then we entered the same search query "Padmini Paladugu" at http://localhost/yioop/ in this case we know the user clicked on the link at http://www.cs.sjsu.edu/faculty/pollett/masters/ and the link exists in the toolbar data. We can see the difference in the results in the given Figure 29 and Figure 30 below.

When the search query "Padmini Paladugu" entered at www.yioop.com.

Figure 29: Tests at Yioop

When the search query "Padmini Paladugu" entered at http://localhost/yioop/

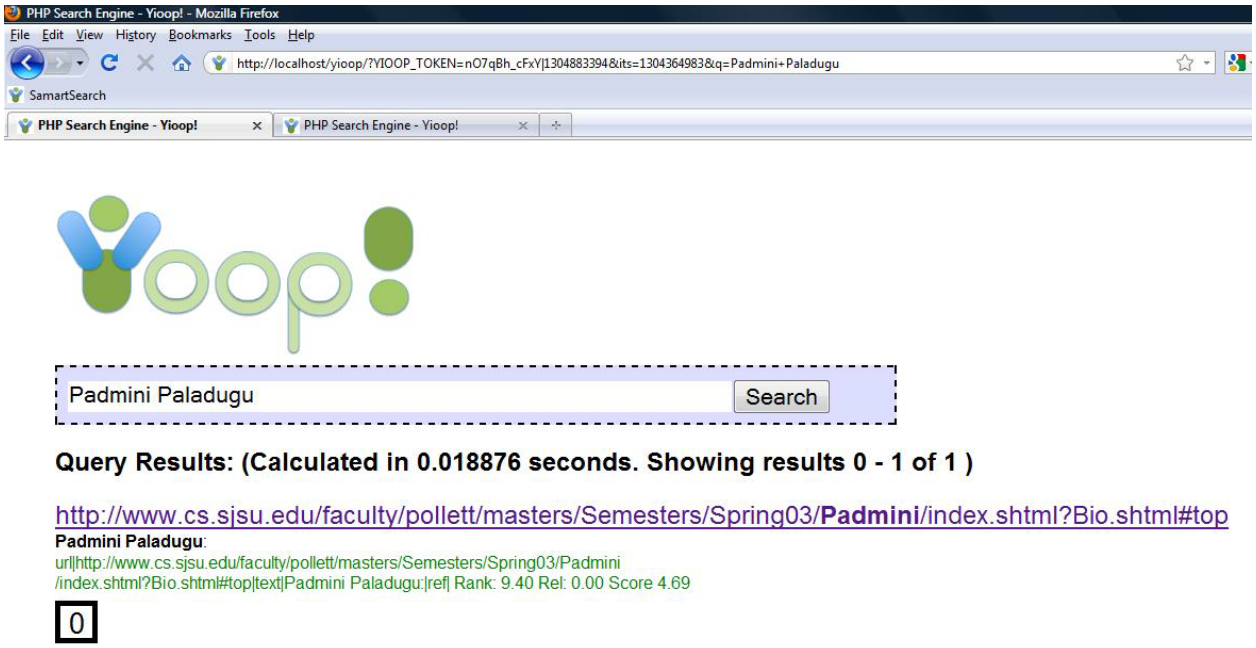

Figure 30: Test at localhost Yioop

In this test also we can observe that the result is given the Rank 9.40.

Test 3

This test is to compare the search results given for the same search query at www.yioop.com and the tool bar installed http://localhost/yioop/. The tool bar data contains the user captured data from the url http://www.cs.sjsu.edu/faculty/pollett/masters/.

When the search query “Vijaya Pamidi” query is given at http://localhost/yioop/

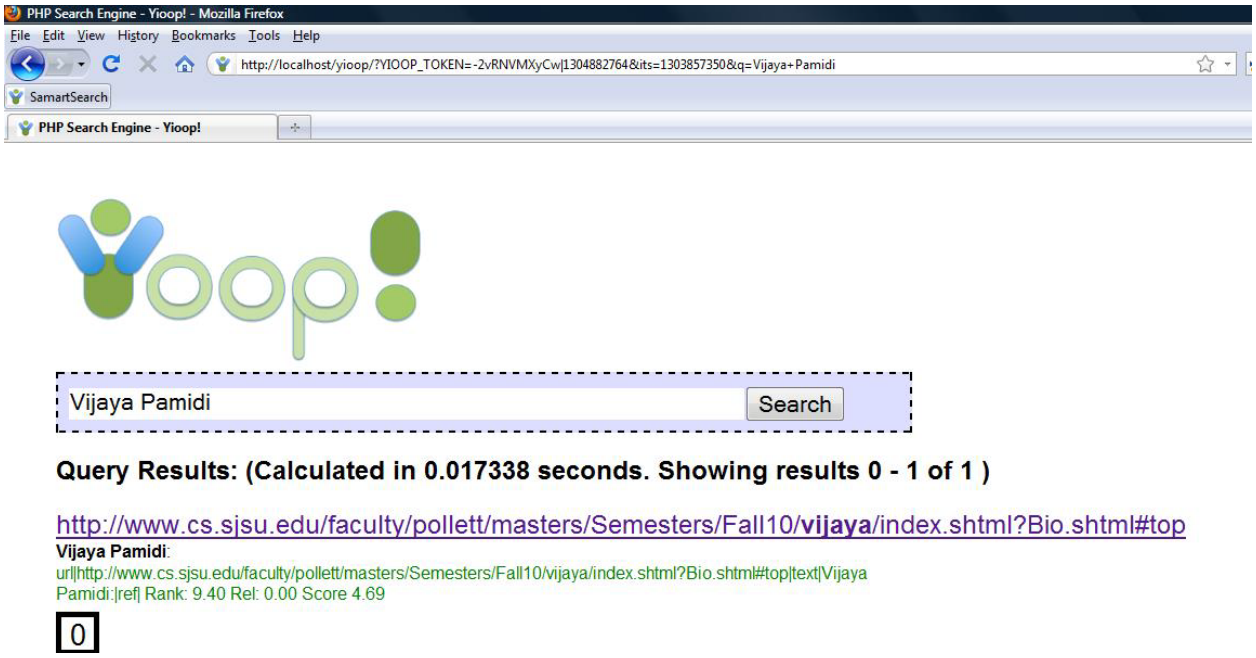

Figure 31: Test at localhost

When the same query is given at www.yioop.com

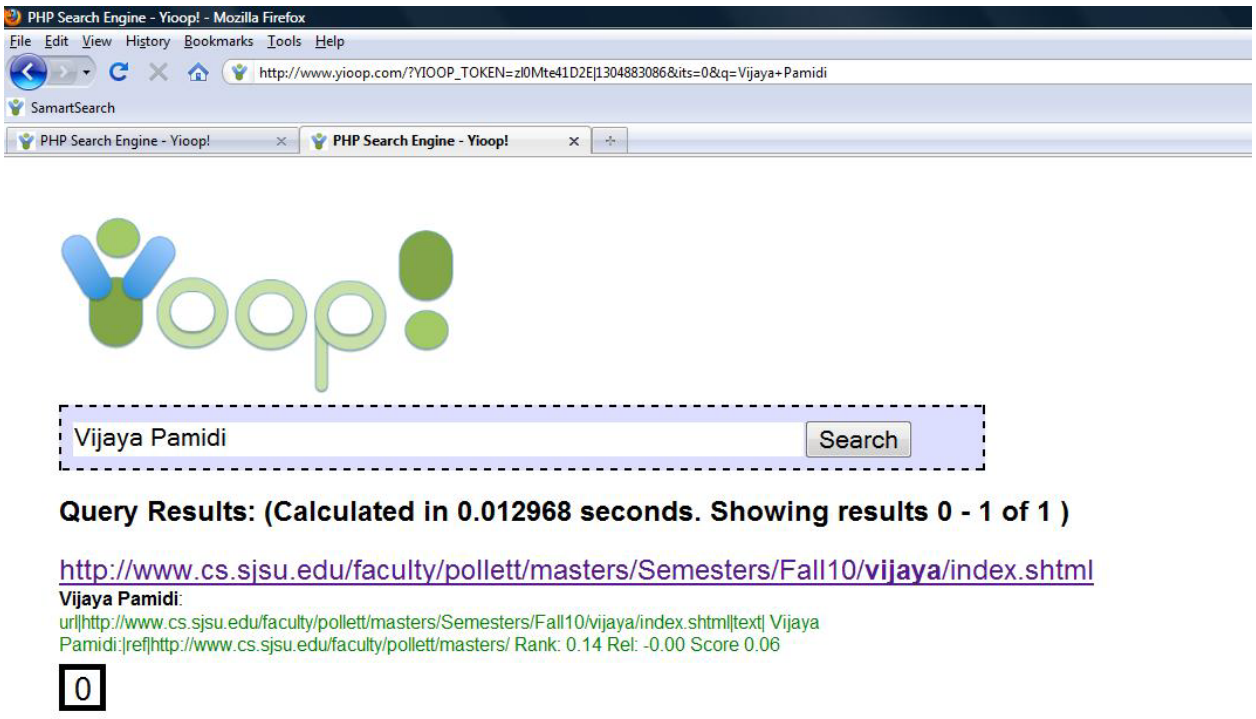

Figure 32: Test at Yioop

From these two tests the returned results url is same but we can observe the difference in the Rank is 0.14 for the www.yioop.com and the Rank is 9.40 for the http://localhost/yioop/. This variation in the ranks exists as the Yioop results are ranked by the links of the web pages that are crawled with open web and whereas the rank for the results from localhost Yioop is based on the particular data that gives the page a high rank.

Test 4

In test4 we would like to test the relevance of the results returned for the same search query. For this we selected the search query “Silver”. We gave the same query in the Yioop and the local host. In this case as the user was asked to browse the ajpm-gold web site the more relevant result would be the link pointing to that web site. When the test was conducted the results given in both the cases is shown in Figure 33 and Figure 34 respectively.

Figure 33: Test4 testing the relevance of results

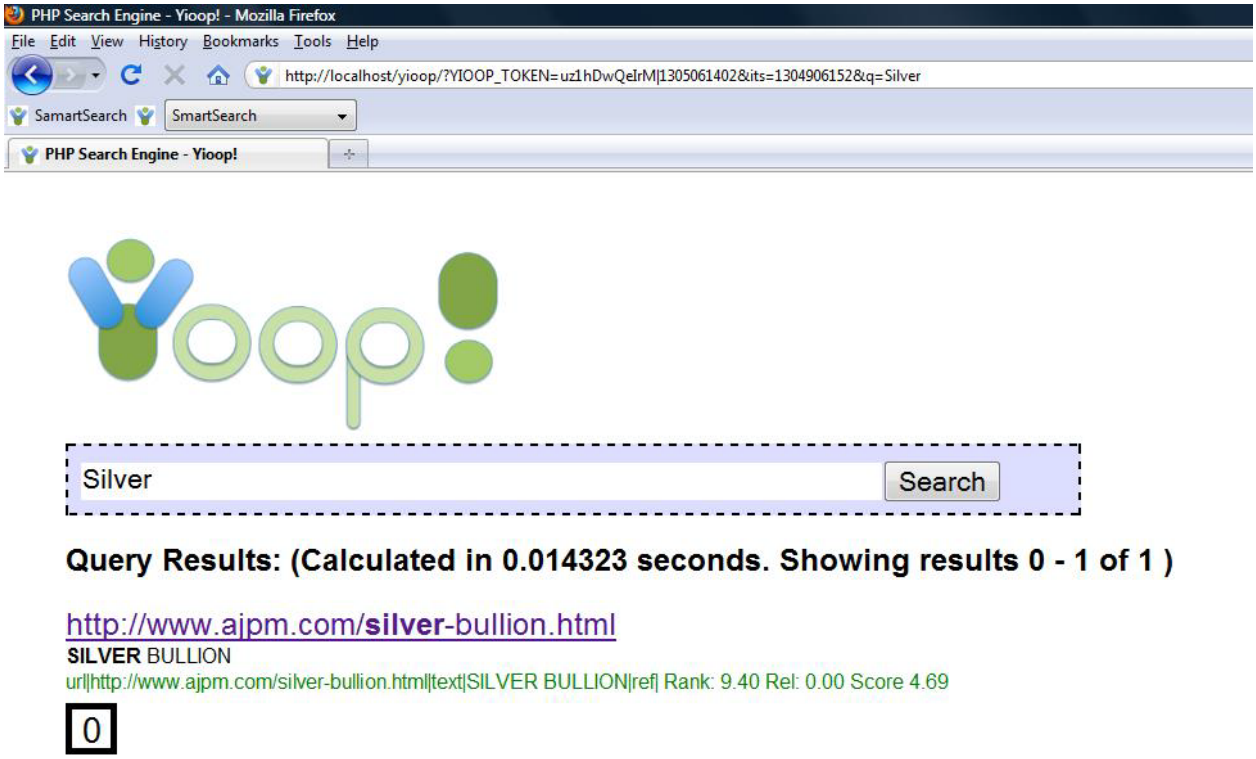

Figure 34: Test4 testing the relevance of results

Observations

The first observation from test 1 is that the toolbar data is getting crawled by the Yioop server and returning the results. From test 2 the second observation is toolbar data is obviously helping to return the results which Yioop is giving 0 results for a search query. With test 3 we can observe that the ranking given by the Yioop with toolbar data is high. The Yioop search engine's ranking is based on the link of the web pages that Yioop crawled with the open web. The toolbar data sent to Yioop contains the user clicked links, the url user clicked, target url, timestamp and language. These details should help a particular url get a high rank compared to the rank the same url was getting with present Yioop search. This can be observed from test 3 as the result returned for the search query given has the rank 0.14 and for the same search query with the toolbar data test returned the same url in the search result but got a high rank of 9.40. The score of the Yioop is 0.06 for the url returned and where as the score for the same results with toolbar data is 4.96.

These tests conducted show that user having the Smart search toolbar Firefox extension provide a vast improvement to search for getting user centric search results and it is an added advantage for the Yioop users.

11. Conclusion

The main goal of the Smart search Firefox add on is to return the user centric search results from which user gets better search results compared to the current once from the www.yioop.com. The user end functionalities of the toolbar extension, capture of the user clicks and the other related information when user clicks on a particular link on a web page are all achieved using the Smart search add on. Synchronization of data transfer from toolbar to Yioop to the target folder and the toolbar data crawl by Yioop is done efficiently. After the crawl by the queue server is completed then the results are returned for a particular search query entered by the user. From the tests performed it was shown that having the Smart search toolbar extension improvised the search experience for the users of Yioop search engine by getting better and more user centric search results.

When the user is installing the Smart search engine one should be aware of the advantages of this smart search toolbar for Yioop search engine and how this is advantageous to the user. More over the user should also be made aware that the most of the web browsing and links clicked by the user would be captured by the toolbar.

References

[1] http://en.wikipedia.org/wiki/XUL

[2] http://en.wikipedia.org/wiki/PHP

[3] http://en.wikiversity.org/wiki/Google_Matrix

[4] http://www.seekquarry.com/

[5] https://developer.mozilla.org/en/Storage

[6] https://developer.mozilla.org/en/Building_an_Extension

[7] Google's Page Rank and Beyond: The Science of Search Engine Rankings by Amy N. Langville and Carl D. Meyer- 2006.

[8] Building Social Web Applications: by Gavin Bell. O'Reilly Media. 2009.

[9] Programming Firefox: Building Rich Internet Applications with XUL: by Kenneth C. Feldt. O'Reilly. 2007.

[10] Building JavaScript- Complete: by Steven Holzner. 1998

[11] http://developer.mozilla.org/en/docs/Building_an_Extension":Official page of Mozilla.

Article References:

[1] Konstantin Avrachenkov and Nelly Litvak. The effect of new links on Google Page Rank. Technical report, INRAIA,
July 2004

[2] Matthew Richardson and Pedro Domingos. The Intelligent Surfer: Probabilistic Combination of Link and Content
Information in Page Rank. Advances in Neural Information Processing Systems, 14:1441-8, 2002.

[3] Taher H. Haveliwala (1999). Efficient computation of Page Rank. Technical report, Stanford University, Stanford,CA.