Fall 2012

Extending Yioop! Abilities to Search the Invisible Web

Tanmayee Potluri
San Jose State University

Extending Yioop! Abilities to Search the Invisible Web

A Writing Project

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 Computer Science

by

Tanmayee Potluri

December 2012

SAN JOSÉ STATE UNIVERSITY

The Undersigned Writing Project Committee Approves the Writing Project Titled

Extending Yioop! Abilities to Search the Invisible Web

by

Tanmayee Potluri

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Chris Pollett, Department of Computer Science 11/20/2012

Dr. Sami Khuri, Department of Computer Science 11/20/2012

Frank Butt, Department of Computer Science 11/20/2012

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my gratitude to Dr. Chris Pollett for being my advisor and guiding me throughout my project. I would also like to thank my committee members Dr. Sami Khuri and Prof. Frank Butt for accepting to be in the committee and providing me necessary inputs regarding the project. Also, I am highly indebted to my family and friends for their constant support throughout the project. I would also like to convey my special thanks to my friends who have helped me in extensive usability testing.

ABSTRACT

EXTENDING YIOOP ABILITIES TO SEARCH THE INVISIBLE WEB

Yioop is an open source search engine written in PHP. Yioop can be used for personal crawls on predefined URLs or as any traditional search engine to crawl the entire Web. This project added to the Yioop search engine the ability to crawl and index various resources that could be considered a part of the Invisible Web. The invisible web refers to the information like database content, non-text files, JavaScript links, password restricted sites, URL shortening services etc. on the Web. Often, a user might want to crawl and index different kinds of data which are commonly not indexed by the traditional search engines. Mining of log files and converting them into a readable format often helps in system management. In this project, the file format of log files has been studied and was noticed that they contain some predefined fields and a user is provided with a user interface to provide details of field names and the field types of the log fields. Indexing databases is one of the other features that would be helpful to a user. This project will act as a resource to the user to index the database records by entering a simple query into the interface created in Yioop and the specific database records are crawled and indexed providing the user with the ability to search for his desired keywords. This project's goal was to successfully embed these features in Yioop using the existing capabilities of Yioop.

Table of Contents

List of Figures 7

List of Tables 7

1 Introduction 8

2 Preliminaries 10

  2.1 Yioop! 10

    2.1.1 Simple crawl in Yioop 11

    2.1.2 Archive Crawling in Yioop! 12

    2.1.3 Architecture 14

  2.2 Invisible Web 16

    2.2.1 JavaScript links 16

    2.2.2 URL Shortening 17

    2.2.3 Log Files 18

    2.2.4 Database Records 19

3 Design 20

  3.1 User Interface Design 21

    3.1.1 Log Files 23

    3.1.2 Database Records 29

  3.2 Backend Design 30

    3.2.1 URL Shortening 31

    3.2.2 Log Files 31

    3.2.3 Database Records 33

4 Implementation 33

    4.1 URL Shortening Links 34

    4.2 Indexing Log Files in Yioop! 37

    4.3 Indexing Database Records in Yioop! 41

5 Testing and Results 45

  5.1 URL Shortening 45

  5.2 Usability testing 47

  5.3 Testing efficiency 55

6 Conclusion 59

7 References 61

List of Figures

Figure 1: Yioop Web User Interface.......... 11
Figure 2: Recrawl Options Interface for Archive Crawling.......... 14
Figure 3: Format of a line in a log file.......... 19
Figure 4: Database Table.......... 20
Figure 5: Manage Crawls Interface.......... 22
Figure 6: Web Crawl options tab.......... 22
Figure 7: Archive Crawl options tab.......... 23
Figure 8: User Interface for Log Files Crawling.......... 24
Figure 9: List of field types that can be specified.......... 25
Figure 10: Steps to enter details for crawling the log files.......... 28
Figure 11: Databases Interface.......... 29
Figure 12: Database connection details entered.......... 30
Figure 13: Process design for archive iterator in Yioop.......... 32
Figure 14: Sequence of steps in the design flow of crawling and indexing log files.......... 32
Figure 15: Sequence of steps in the design flow of crawling database records.......... 33
Figure 16: Search results for Yahoo after the crawl.......... 35
Figure 17: 301 redirect to the Yahoo link.......... 35
Figure 18: Results shown before and after the code was added.......... 36
Figure 19: nextPage function for creating HTML pages.......... 39
Figure 20: Search results obtained after crawling log records.......... 40
Figure 21: An open log record.......... 41
Figure 22: Search for all log records with a 302 status code.......... 41
Figure 23: Database connection details.......... 44
Figure 24: Search results for databases.......... 44
Figure 25: Results before the addition of code.......... 46
Figure 26: Results after the addition of code.......... 46
Figure 27: Time taken for each user to complete a task.......... 50
Figure 28: Graph displaying the average time taken by users for each of the tasks.......... 51
Figure 29: Accuracy Percentage of tasks performed by user.......... 52
Figure 30: Graph showing the amount of time taken by users for the first time and second time.......... 53
Figure 31: Average rating of satisfaction of user.......... 54
Figure 32: Straight line graph of time versus the number of log records.......... 56
Figure 33: Crawl results for second query.......... 57
Figure 34 : Comparison of Yioop with Mysql LIKE operator.......... 58

List of Tables

Table 1 : Assumed number of mistakes for each task.......... 52

1 Introduction

A search engine is often used to search for information about a particular keyword or a topic in general known as a search query [16]. A search engine then retrieves all the results matching that particular query and often, these results are web pages [16]. The two main concepts behind search engines are crawling and indexing. Crawling is the process of fetching pages or data and indexing is the process of storing the data in a processed format and provide necessary help to rank the importance of the content [15]. Yioop is one such open-source search engine written in PHP [1]. Yioop can be used for personal crawls on the Web and also can be used to crawl and index different kinds of data not usually crawled by traditional search engines [1]. The project added to Yioop the ability to crawl and index a part of the invisible web.

Yioop has the ability to crawl and index different file formats. This is called an archive crawl in Yioop terminology. This ability allows Yioop to crawl different file formats. Code written for a specific file format and code to iterate over this format are put in archive bundle iterators in Yioop. Using this archive crawling set up, Yioop was extended to crawl log files and databases in this project. Code to allow Yioop to deal with short URLs was also added.

A log file is typically a file to which a computer system writes a record of its activities [9]. Log files are automatically created and maintained by a server containing the information of the activity performed by it [9]. An example of a log file is an Apache access log file which maintains a history of page requests [9]. With the increase in science and technology, computing systems are becoming more complex as a result of the varied software and hardware components [2]. Analysis and mining of log files plays an important role in managing and monitoring these systems [2]. For this reason, a user often might to analyze a log file, search for an activity, search for successful requests, etc. AWStats, Webalizer, W3Perl and Analog are few of the log file analyzers available in order to help the users check the statistics of the log files. AWStats is a powerful tool that generates advanced log file statistics and provides all possible statistical information about log files in the form of graphical web pages [10]. The kind of data provided by AWStats is the number of visits, number of unique visitors, visits duration and last visits, authenticated users, and last authenticated visits, browsers, HTTP codes, domains, etc [10].

While it is useful to know the statistical information about the log files, it is also important to search for a particular activity or a request or a keyword in log files. A search engine to crawl log files will help in this regard. In this project, customized crawling of these log files was made possible in Yioop. A user is given an interface in Yioop to provide his own details of the fields in the log records. The type of the field and also the field names can be specified by the user. Web pages are built in the archive crawling process performed by Yioop. The building of these web pages and indexing is done in the archive bundle iterator written for log files. Using AWStats, statistical information about log files is generated whereas using the ability in Yioop to crawl and index log files, one can search for a particular request or activity in log files using a search interface.

A database is a collection of records which can contain any kind of data [12]. There can be cases when a user wants to search for a specific word in a database. Database indexing is often helpful in analyzing the data available in a database and search for information in a database. Sphinx is one of the open-source search engines designed for indexing database content specifically [8]. It performs indexing on the database file provided to it and returns search results for a particular search keyword. It provides details of the record where the particular search keyword is present. Sometimes, the user might want to limit the record search according to a particular query. A similar kind of functionality is added to the Yioop search engine in this project. The user is given the opportunity to enter the database connection details and also a query. Based on the query entered by the user, the number of records to be crawled and indexed are filtered and indexed by Yioop. The user can then search for any keyword in the indexed data.

We now discuss the organization of the rest of this report. The next section provides insights into preliminaries of the project which are necessary in order to understand this project. Later, the design of the project is described which includes both the user interface design and also the backend design. Then the implementation details of the project are presented. Finally, a brief conclusion is given.

2 Preliminaries

In order to understand this project and its inner workings, a basic understanding of some concepts becomes important. A brief introduction and explanation of these concepts has been provided in this section. This section introduces Yioop and its web and archive crawling concepts. It also provides a brief introduction to the architecture of Yioop. Following the introduction to Yioop, this section also introduces the concepts of invisible web and explains various resources of invisible web. These resources include the file formats crawled and indexed in this project.

2.1 Yioop!

Yioop! is an open source search engine written in PHP by Dr. Chris Pollett [1]. Yioop! can be used both as a traditional search engine or it can be modified to act as a search engine only for a predefined set of domains or URLs [1]. Yioop is licensed under GPLv3 and SeekQuarry is the parent site for Yioop [17]. Yioop has been designed in order to ease the usage of personal crawls, support distributed crawling of the web, make it easier to crawl archives and perform archive crawls [17]. The user has complete control over the indexes of the crawls performed in Yioop. The user can limit the search results based on his requirement.

The web user interface of Yioop looks similar to a traditional search engine and is used to enter search queries into it. The user interface looks as shown below in the figure:

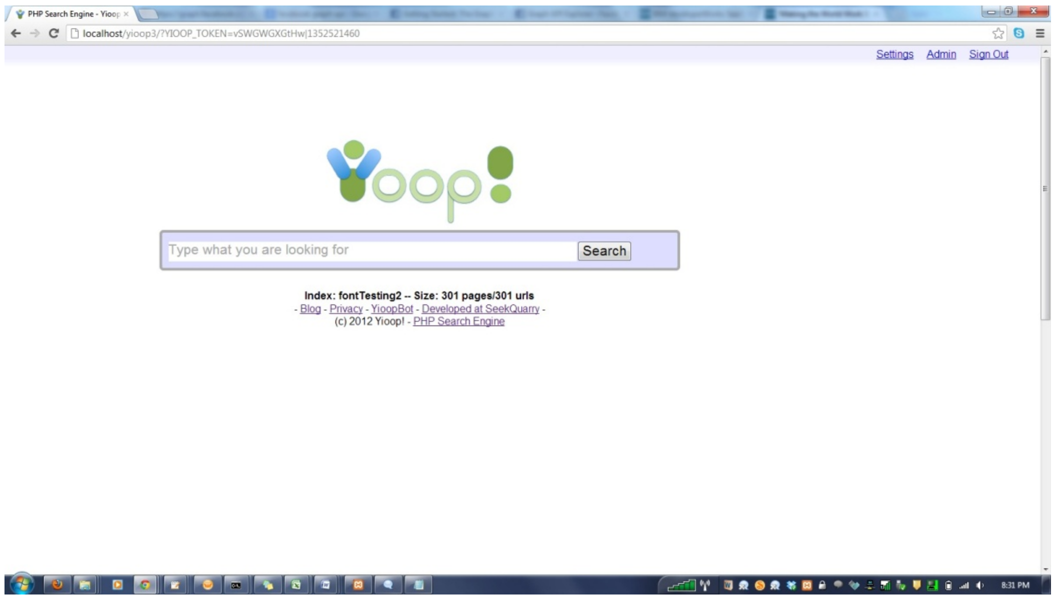

PHP Search Engine - Yioop
localhost/yioop3/?YIOOP_TOKEN=vSWGWGXGtHw|1352521460
Settings
Admin
Sign Out
Yioop!
Type what you are looking for
Search
Index: fontTesting2 -- Size: 301 pages/301 urls
- Blog - Privacy - YioopBot - Developed at SeekQuarry -
(c) 2012 Yioop! - PHP Search Engine
8:31 PM

Figure 1: Yioop Web User Interface

2.1.1 Simple crawl in Yioop

For crawling and indexing web pages or any kind of file formats in Yioop, knowledge of performing a simple crawl is necessary. In order to perform a simple crawl in Yioop, Yioop needs to be downloaded and installed on the system. Xampp should also be downloaded and installed on the system which provides Apache, PHP and MySql in order get Yioop to work. For easier access, Xampp should be installed under the C:/ drive and Yioop should be extracted and placed in the htdocs folder of Xampp. Yioop can be installed both on Linux or Windows 7 systems.

After the successful installation of Yioop and Xampp on the system, a onetime configuration of Yioop should be done [1]. The data folder for the crawls is created under the htdocs folder in this step. This will hold the data that is indexed by Yioop. All the steps necessary to perform a simple crawl in Yioop have been provided in Yioop's documentation in SeekQuarry. The seed sites available in the web crawl options interface of Yioop can be used to perform an initial crawl in Yioop. In order to perform a simple crawl in Yioop, two programs need to be started from the command line or using the web interface of Yioop. They are fetcher.php and queue_server.php. The queue_server program maintains a queue of URLs that are going to be scheduled and also keeps track of what has been seen and robots.txt info [1]. This program also performs the responsibility to create the index_archive that is used by the search interface in the front end of Yioop [1]. The fetcher program downloads batches of URLs provided by the queue_server [1]. A name can be given to the crawl and it can be set as index to be searched. In order to test the code written for this project and also crawl the log files and databases, it is important to know how a simple crawl is performed in Yioop and how the data needs to be indexed.

2.1.2 Archive Crawling in Yioop!

A major part of this project is connected to archive crawling in Yioop. Archive crawling in Yioop is used to perform crawling of the previous crawls performed in Yioop or crawl one of the web archive file formats like Arc, MediaWiki XML, ODP RDF [1]. In order to crawl each of these file formats, separate archive bundle iterators have been written in Yioop.

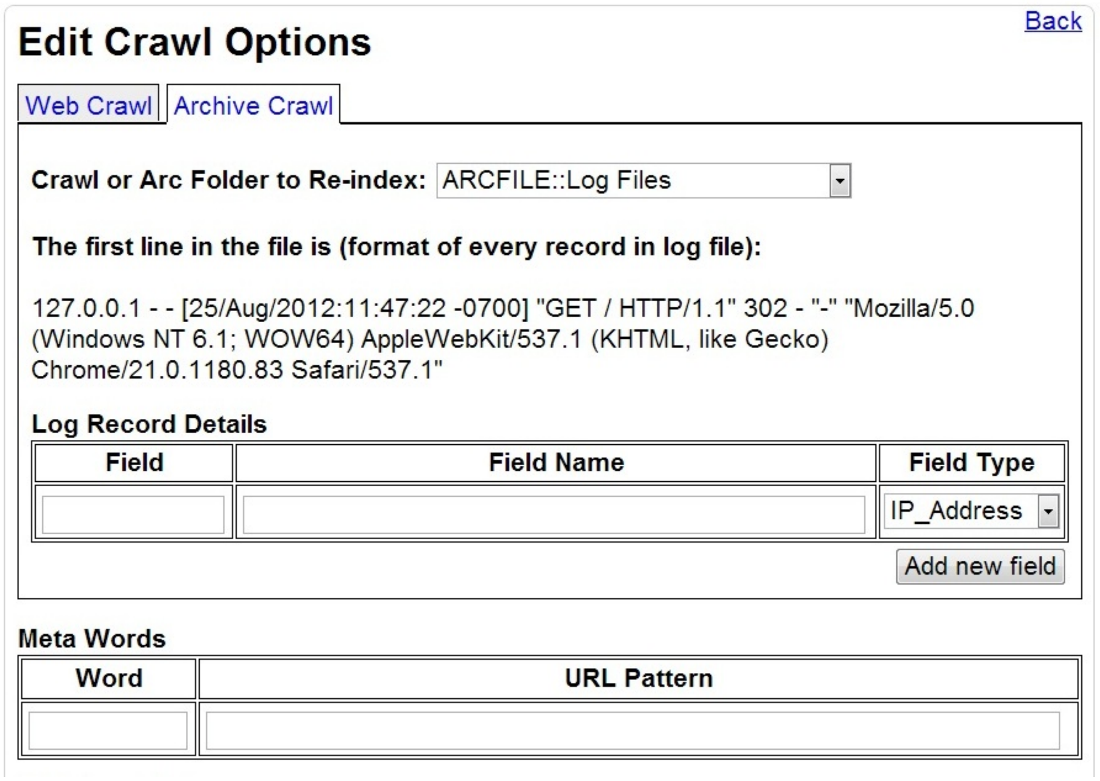
In order to perform an archive crawl in Yioop, one has to create a folder "archives" in the cache folder of the WORK_DIRECTORY of Yioop. In the archives folder, another folder should be created to store all the files of a particular format that are to be crawled. As an example, in order to perform a crawl on the log files, one needs to create a folder like my_log_files in the WORK_DIRECTORY/cache/archives folder and place all the .log files to be crawled in there.

Each of those folders in archives folder should contain a file named arc_description.ini which contains the text like,

```
arc_type = 'LogArchiveBundle';

description = 'Log Files';
```

Once this folder is created ARCFILE::Log Files is seen as one of the options to recrawl. The arc_type should be given as LogArchiveBundle, DatabaseArchiveBundle etc. The description can be given as per the wish of the user and it is displayed on the web user interface.



Edit Crawl Options

Crawl or Arc Folder to Re-index: ARCFILE::Log Files

The first line in the file is (format of every record in log file):

```
127.0.0.1 - - [25/Aug/2012:11:47:22 -0700] "GET / HTTP/1.1" 302 - "-" "Mozilla/5.0
(Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML, like Gecko)
Chrome/21.0.1180.83 Safari/537.1"
```

Log Record Details

Field	Field Name	Field Type
		IP_Address

Meta Words

Word	URL Pattern

Figure 2: Recrawl Options Interface for Archive Crawling

In this project, two archive bundle iterators had to be developed in order to perform crawls on log files and databases. Therefore, these crawls were in the form of archive crawls. The archive iterators are LogArchiveBundleIterator and DatabaseArchiveBundleIterator. A brief description of log files and databases is provided in the later part of this section.

2.1.3 Architecture

The goal of this project is to add to the Yioop search engine the ability to crawl and index log files and database records. In order to accomplish these tasks, it is useful to know the architecture of Yioop and how it needs to be modified to implement these features.

Yioop is designed using the web-based Model-View Controller framework [1]. A server like Apache and PHP 5.3 or later are the main components required in order to install Yioop [1].

Both of these are available in Xampp. PHP multi-curl libraries are required for Yioop in order to download the web pages [1]. Yioop can be configured to have multiple queue servers and fetcher in order to perform crawls. This is made possible in Yioop using the simple distributed model [1].

There are three major scripts in Yioop: namely, queue_server.php, fetcher.php and index.php. The queue_server program maintains a queue of URLs that are going to be scheduled and also keeps track of what has been seen and robots.txt info [1]. This program also performs the responsibility to create the index_archive that is used by the search front end [1]. The fetcher program downloads batches of URLs provided by the queue_server. the index.php page acts as the search engine web page [1].

In order to implement this project, few of the existing files in Yioop needed to be modified and also new files needed to be added into the Yioop's architecture. A brief explanation of these files and folders has been provided in this section.

The lib folder has the archive bundle iterators folder which contains all the archive bundle iterators needed to crawl and index different file formats in Yioop. The two archive bundle iterators namely, LogArchiveBundleIterator and DatabaseArchiveBundleIterator to crawl and index the log files and databases respectively need to be added into this folder of Yioop. The controllers folder contains all the controllers used by the Yioop search engine. The admin_controller file in this folder of Yioop needs to be modified in order to build a user interface in Yioop.

The css folder in Yioop contains the stylesheets which have all the css styles defined for all elements of html used in Yioop [1]. The search.css file needs to be modified in order to define styles for all the html elements introduced into Yioop for the user interfaces of log files and databases. There are also other folders present in Yioop which perform the necessary functionalities in Yioop. In addition to all these folders, Yioop also has a WORK_DIRECTORY which is used to store the pages crawled in Yioop and also the search results of Yioop [1].

2.2 Invisible Web

This subsection provides a brief introduction to the field of invisible web and some insights into few resources that constitute the deep web. The resources mentioned here constitute the ones implemented in this project and also few of them which were implemented to a considerable extent.

Invisible web also known as deep web, deep net or hidden web is that part of the web which is not indexed and is not a part of the surface web [4]. The invisible web is a lot more larger in magnitude when compared to the surface web [4]. The invisible web may constitute many resources like shortened links, database content, any content in specific file formats, JavaScript links, password protected sites etc which are generally not indexed by the traditional search engines.

In this project, log files and databases are made to crawl and index in Yioop. Preliminary work has been done in order to extract JavaScript links in web pages. A brief introduction to all these resources of Invisible Web are explained in this section.

2.2.1 JavaScript links

JavaScript content constitutes a major part of the web pages on the Web. There are few pages in the Web which are accessible only through links in the JavaScript content [4]. These links are dynamically created in the JavaScript files and therefore are not crawled and indexed by traditional search engines. The challenge in accessing these links lies in the fact that they are dynamically created by the code in the JavaScript. In order to extract such links, a JavaScript parser needs to be created and the functions in the JavaScript need to be implemented.

In this project, there was some research work done in this area. The simple URLs in JavaScript content was extracted. The JavaScript content could be parsed using Spider Monkey (a JavaScript engine) but Spider Monkey did not contain all the functions to convert a dynamic link in JavaScript to a complete URL. As a result of this, it was a challenge implementing all the JavaScript functions in PHP and trying to embed such functions in Yioop. Therefore, this feature could not completely implemented in Yioop and can be included as an improvement for the project.

2.2.2 URL Shortening

URL shortening is a technique on the Web in which a URL is made short in length and still redirected to the required page [14]. The redirection is achieved using an HTTP Redirect on a short domain name which in turn links to the web page that has a longer URL [14]. For example, the URL http://www.yahoo.com is shortened to bitly.com/4bYAV2. There are many URL shortening services of which bit.ly and tinyurl are few of them.

URL shortening is convenient for Twitter and Identi.ca in which the number of characters are limited a particular number of characters [14]. The underlying address can be disguised using URL shortening services [14]. URL shortening services are one source of dark content on the Web. It is a source of dark content as in ranking results search engines often attribute the link to the URL short link service rather than to where the link points to. In this project, research was done on how Yioop deals with these links and Yioop was made to work correctly with these links. In this regard, a simple crawl was performed on a bitly link alone and the results showed that the crawling and indexing was done on the short link and therefore code was written to handle this behavior and make Yioop to work with the short links correctly.

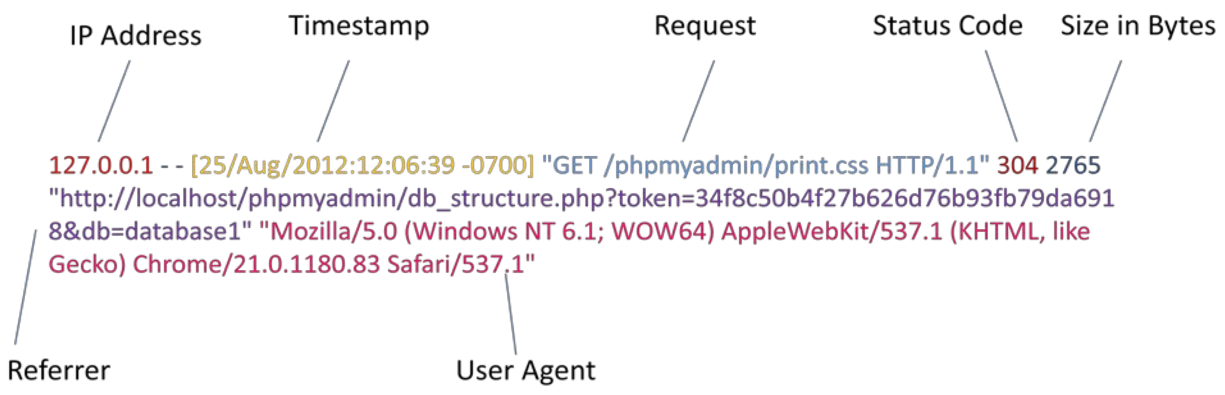
2.2.3 Log Files

Log files are one of those file formats that are not crawled and indexed by traditional search engines. Log files are the files to which a computer system writes a record of its activities [9]. Log files are generally automatically created and maintained by a server containing the information of the activity performed by it [9]. An example of a log file is an Apache access log file which maintains a history of page requests [9]. The format of these log files is standard and is defined by W3C but there can be various other formats also defined by the users [9].

A specific format is defined for Apache log files. This is a common log format that is followed by a number of servers to generate log files. The predefined fields that are present in a log file are:

- IP Address : IP Address is that of the client who sends a request to the server.
- Timestamp : Time when the server is done processing the request
- Request : Request made by the client to the server.
- Status Code : Code sent to client by the server
- Size in Bytes : Size of the object returned by the server to the client
- Referrer : The site that client reports is being referred from.
- User Agent : Information that client refers about itself
- - : Information not returned by the server

An example of a line in an Apache access log file is shown below.



```
127.0.0.1 - - [25/Aug/2012:12:06:39 -0700] "GET /phpmyadmin/print.css HTTP/1.1" 304 2765
"http://localhost/phpmyadmin/db_structure.php?token=34f8c50b4f27b626d76b93fb79da691
8&db=database1" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML, like
Gecko) Chrome/21.0.1180.83 Safari/537.1"
```

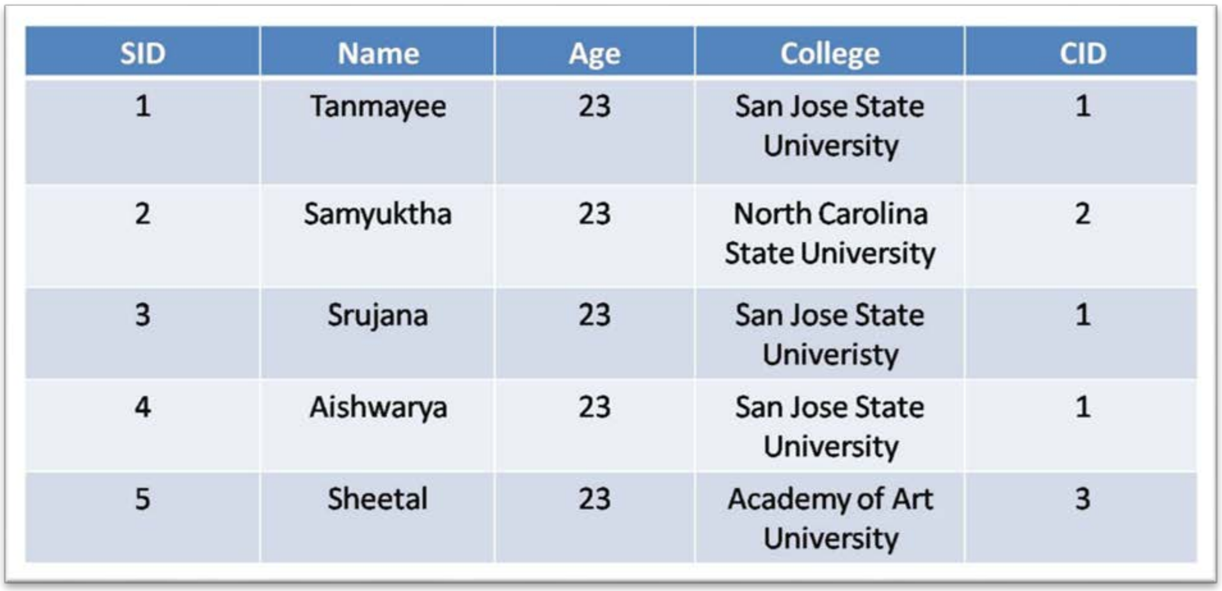
Figure 3: Format of a line in a log file

In this project, these log files have been crawled and indexed by writing an archive bundle iterator. A user interface has been developed for the user to enter their own specified field names for the entries in the log file. The data is then available for search to the user with Yioop.

2.2.4 Database Records

Database files are a collection of similar kind of records [12]. A database can contain any number of tables and tables can contain any number of records. The database content is one kinds of deep web content which is not crawled by traditional search engines.

In this project, a database is crawled and indexed by writing an database archive bundle iterator. The table to be crawled depends on the query specified by the user. The records returned after the query has been executed are indexed. A database table looks as shown below.



SID	Name	Age	College	CID
1	Tanmayee	23	San Jose State University	1
2	Samyuktha	23	North Carolina State University	2
3	Srujana	23	San Jose State Univeristy	1
4	Aishwarya	23	San Jose State University	1
5	Sheetal	23	Academy of Art University	3

Figure 4: Database Table

3 Design

This section describes the design of the various features embedded into Yioop in this project. In order to ensure a smooth flow for the implementation, a good design was necessary before hand. This design helped as a guideline for the implementation of the project. The design section contains only the design flow and steps necessary to follow in order to implement the features. The coding details and the changes made in Yioop's files are described in the implementation section of the report.

The project design consisted of two parts of which one is user interface design and the other is the backend design. User Interface Design consists of the web interface design where the user can enter the details of the field names for log records module and the database connection details for the database module. The backend design consists of the modifications required to be done to the Yioop architecture, the files to be added to Yioop etc. Each of these designs have been explained in detail in the following section.

3.1 User Interface Design

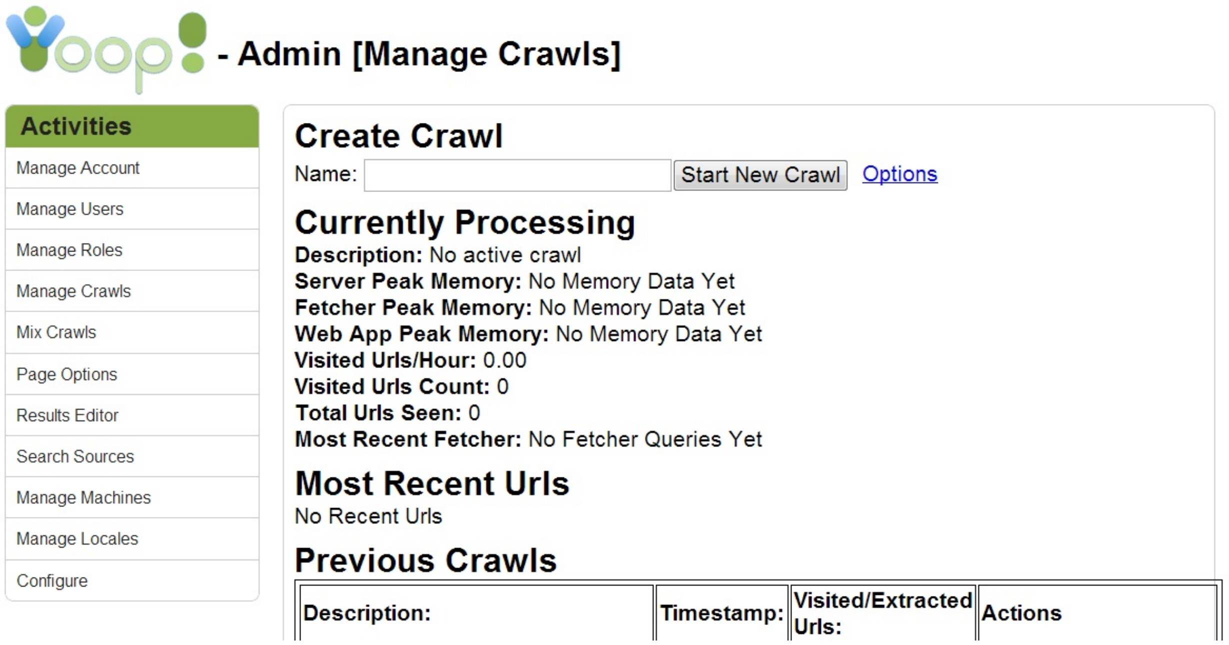
This section describes the user interface design for the crawling of log files and also the database records. The options required for a crawl to be performed should be given by the user prior to starting the crawl. There are two kinds of crawls that can be performed with Yioop.

(1) Web crawl

(2) Archive Crawl

Web crawling refers to the crawling done on the web sites or any domains whereas archive crawling refers to the recrawling of the previously done crawls or crawling any of the archive file formats like arc, MediaWiki XML, ODP RDF, log files, databases etc [1].

The options for a crawl can be specified in the edit crawl options section of Yioop in the web interface. There are two tabs of which one is for web crawls and the other is for archive crawls. The user can navigate to this page from the manage crawls page. The images of manage crawls interface and the edit crawl options interface are shown below.



Create Crawl

Name:

Currently Processing

Description: No active crawl
Server Peak Memory: No Memory Data Yet
Fetcher Peak Memory: No Memory Data Yet
Web App Peak Memory: No Memory Data Yet
Visited Urls/Hour: 0.00
Visited Urls Count: 0
Total Urls Seen: 0
Most Recent Fetcher: No Fetcher Queries Yet

Most Recent Urls

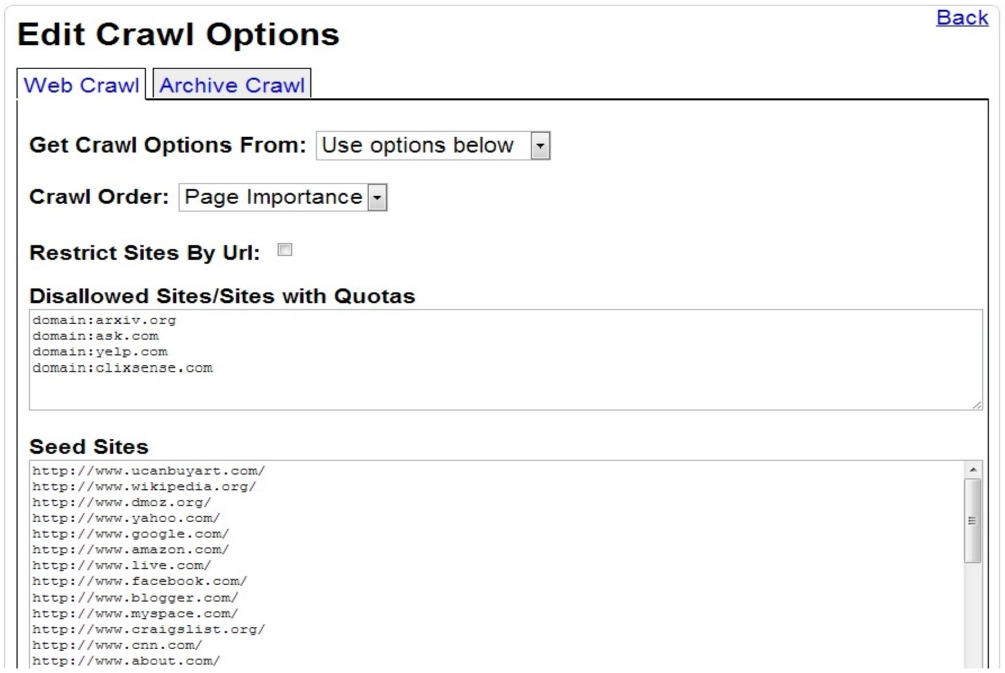
No Recent Urls

Previous Crawls

Description:	Timestamp:	Visited/Extracted Urls:	Actions

Figure 5: Manage Crawls Interface

The details concerned with web crawling can be specified in the web crawl tab of the edit crawl options. The user can specify the seed sites, disallowed sites, any kind of meta words that he wants to add in the web crawl edit options.



Edit Crawl Options

Get Crawl Options From: Use options below

Crawl Order: Page Importance

Restrict Sites By Url: ☐

Disallowed Sites/Sites with Quotas

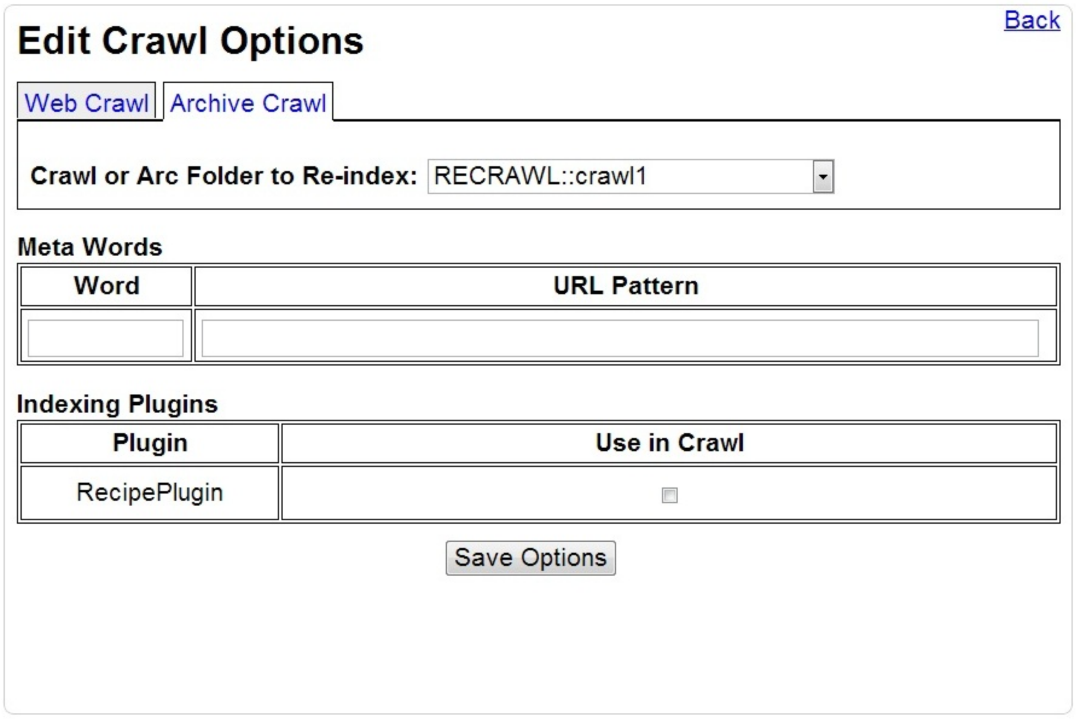
```
domain:arxiv.org
domain:ask.com
domain:yelp.com
domain:clixsense.com
```

Seed Sites

```
http://www.ucanbuyart.com/
http://www.wikipedia.org/
http://www.dmoz.org/
http://www.yahoo.com/
http://www.google.com/
http://www.amazon.com/
http://www.live.com/
http://www.facebook.com/
http://www.blogger.com/
http://www.myspace.com/
http://www.craigslist.org/
http://www.cnn.com/
http://www.about.com/
```

Figure 6: Web Crawl options tab

The details concerned with archive crawling can be specified in the archive crawl tab of the edit crawl options. In this project there are two separate interfaces that are displayed for the log files and databases. The archive tab would show what are the available crawls to recrawl and also the options to crawl different file formats if the user has created the necessary folders.



Edit Crawl Options

Crawl or Arc Folder to Re-index: RECRAWL::crawl1

Meta Words

Word	URL Pattern

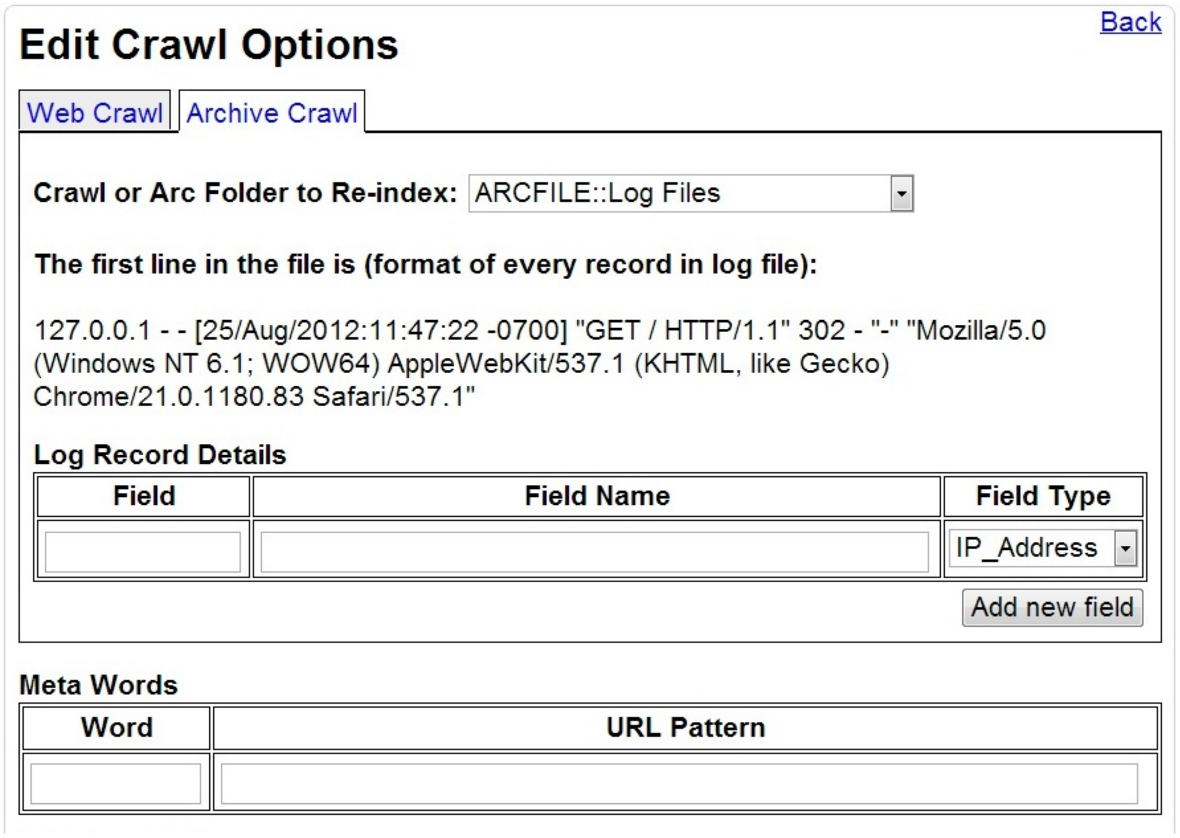
Indexing Plugins

Plugin	Use in Crawl
RecipePlugin	☐

Figure 7: Archive Crawl options tab

3.1.1 Log Files

This subsection describes the user interface design of the log files. Different log files have different formats which are specific to each of them and the files are stored in the system as .log files. The specific file format of each of these different files bring them into the category of archive crawling in Yioop. To provide flexibility to the user in managing, crawling and indexing these log files, a separate user interface has been designed and is displayed to the user when the user selects log files from the crawl options of Yioop. The interface displayed is shown in Figure8.



Edit Crawl Options

Crawl or Arc Folder to Re-index: ARCFILE::Log Files

The first line in the file is (format of every record in log file):

127.0.0.1 - - [25/Aug/2012:11:47:22 -0700] "GET / HTTP/1.1" 302 - "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML, like Gecko) Chrome/21.0.1180.83 Safari/537.1"

Log Record Details

Field	Field Name	Field Type
		IP_Address

Meta Words

Word	URL Pattern

Figure 8: User Interface for Log Files Crawling

The first line of the first log file is displayed as the first line in the interface. An interface is provided to the user to enter his own details regarding the log records. There are three fields that the user needs to enter before saving the crawl options.

Field : This field can be copied from the line displayed to the user.

Field Name : This field is specific to the user. He can enter any name to the field he specified and this name will be attached to the actual field retrieved in the final web pages.

Field type : This specifies the type of field in the log record. There are predefined fields in the drop down box. These fields are IP Address, Timestamp, Request, URL, User Agent, Status Code, Integer. The retrieval of fields from each log record is based on the field types chosen here.

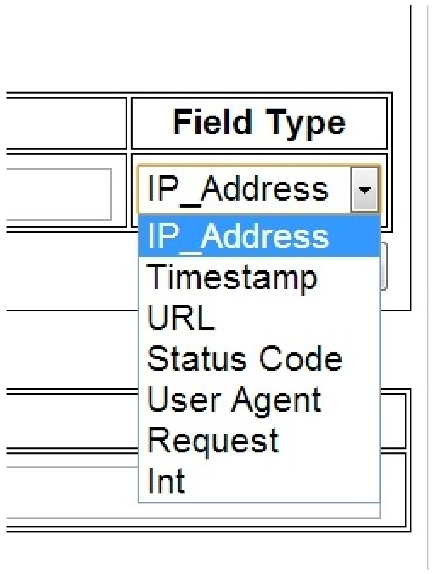
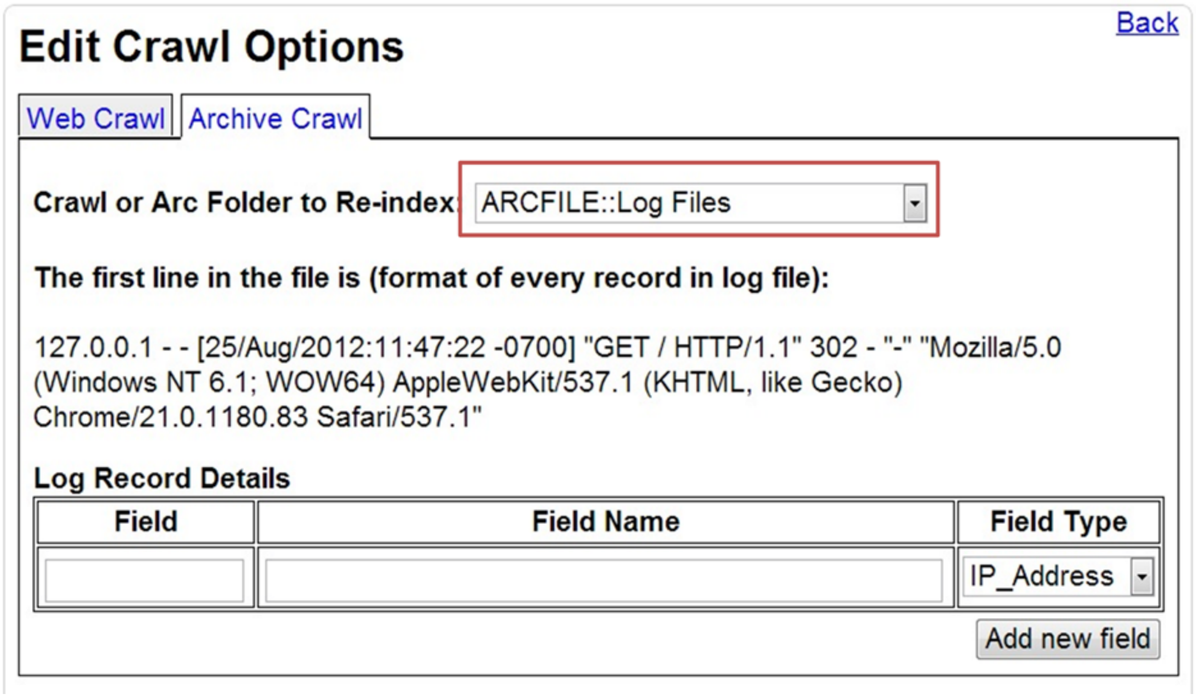


Figure 9: List of field types that can be specified

The "Add New Field" button can be used to add new fields into the interface as needed. The fields entered should be in the order as displayed in the first line of the log file. For example, the IP address is first shown in the line, which means the IP address should be entered as the first field in the interface and next the timestamp. Each line in a log file is treated as a log record. The fields and the field names should be unique for each field of the log record.

The user needs to specify the names for all the fields shown in the first line. A hyphen in the first line implies a missing field, i.e., a field missing in that record but might be present in other records. Any such missing field can be explicitly specified by the user as a separate field provided the user knows what the field is and the placement of the field in the record. It should be placed in the right position where it occurs in the line. After all the fields are specified by the user, the save options button must be clicked in order to save all the details entered. The steps that need to be followed in order to use the interface are shown below.



Back

Edit Crawl Options

Web Crawl | Archive Crawl

Crawl or Arc Folder to Re-index: ARCFILE::Log Files

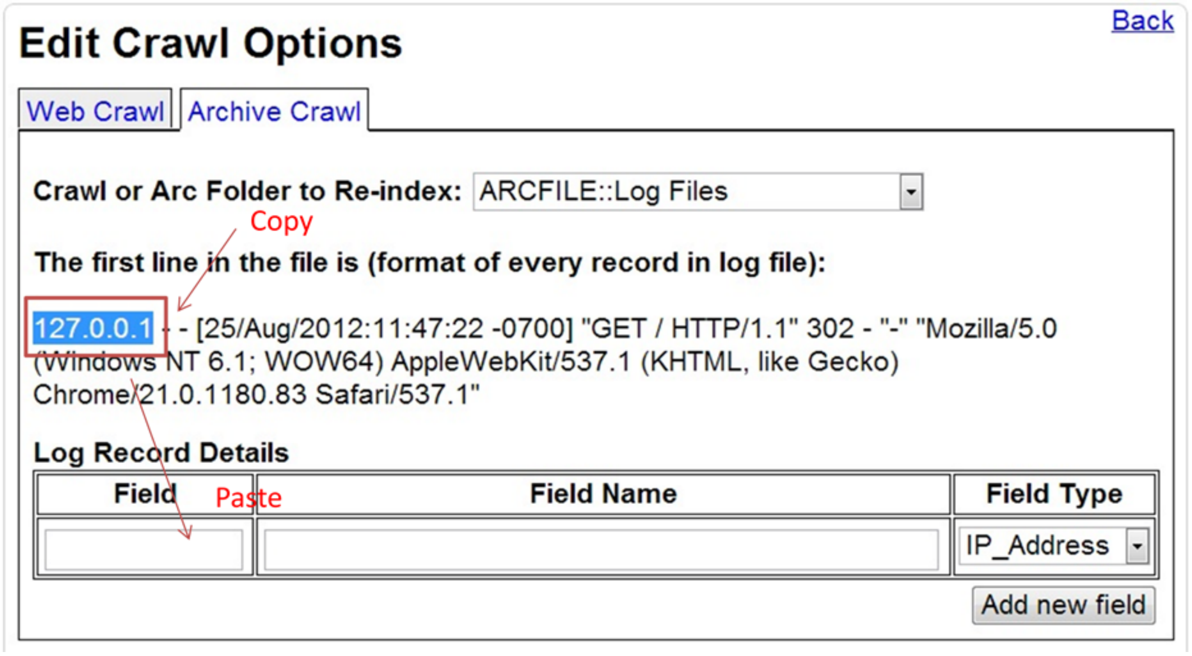
The first line in the file is (format of every record in log file):

127.0.0.1 - - [25/Aug/2012:11:47:22 -0700] "GET / HTTP/1.1" 302 - "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML, like Gecko) Chrome/21.0.1180.83 Safari/537.1"

Log Record Details

Field	Field Name	Field Type
		IP_Address

Add new field



Back

Edit Crawl Options

Web Crawl | Archive Crawl

Crawl or Arc Folder to Re-index: ARCFILE::Log Files

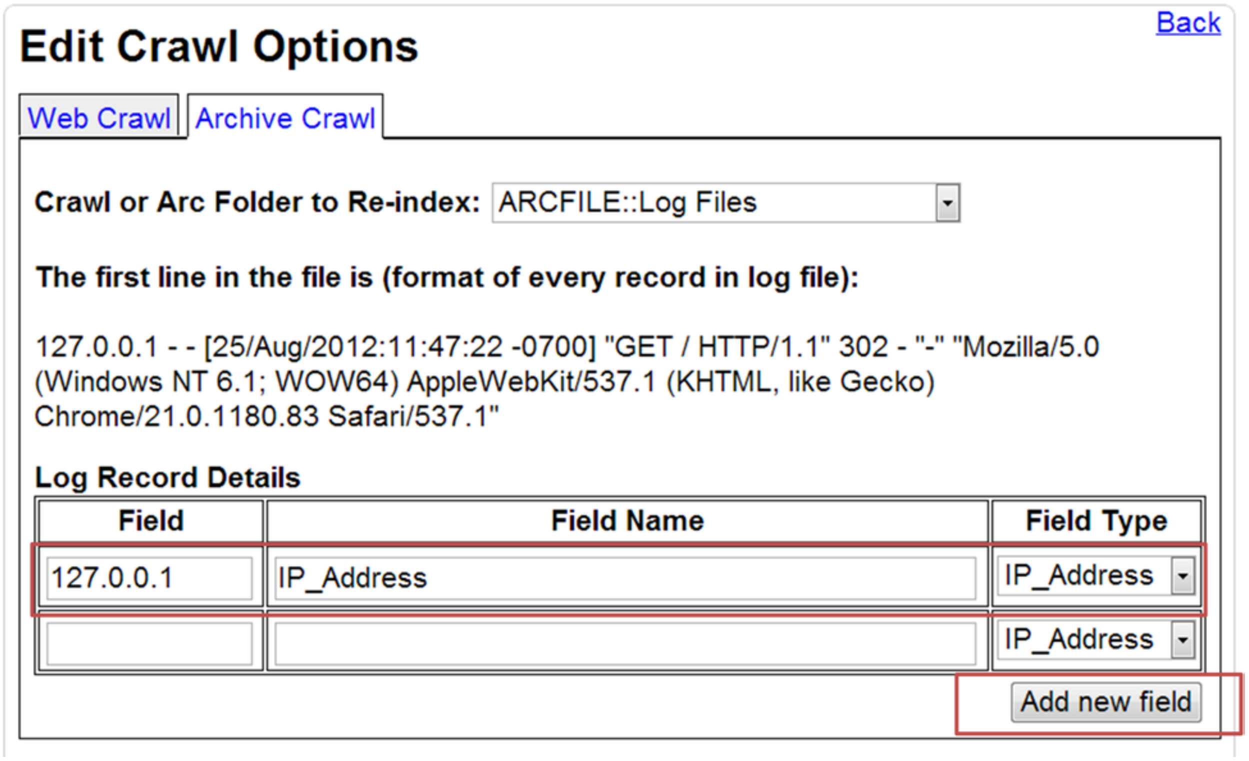
The first line in the file is (format of every record in log file):

127.0.0.1 - - [25/Aug/2012:11:47:22 -0700] "GET / HTTP/1.1" 302 - "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML, like Gecko) Chrome/21.0.1180.83 Safari/537.1"

Log Record Details

Field	Field Name	Field Type
		IP_Address

Add new field



Back

Edit Crawl Options

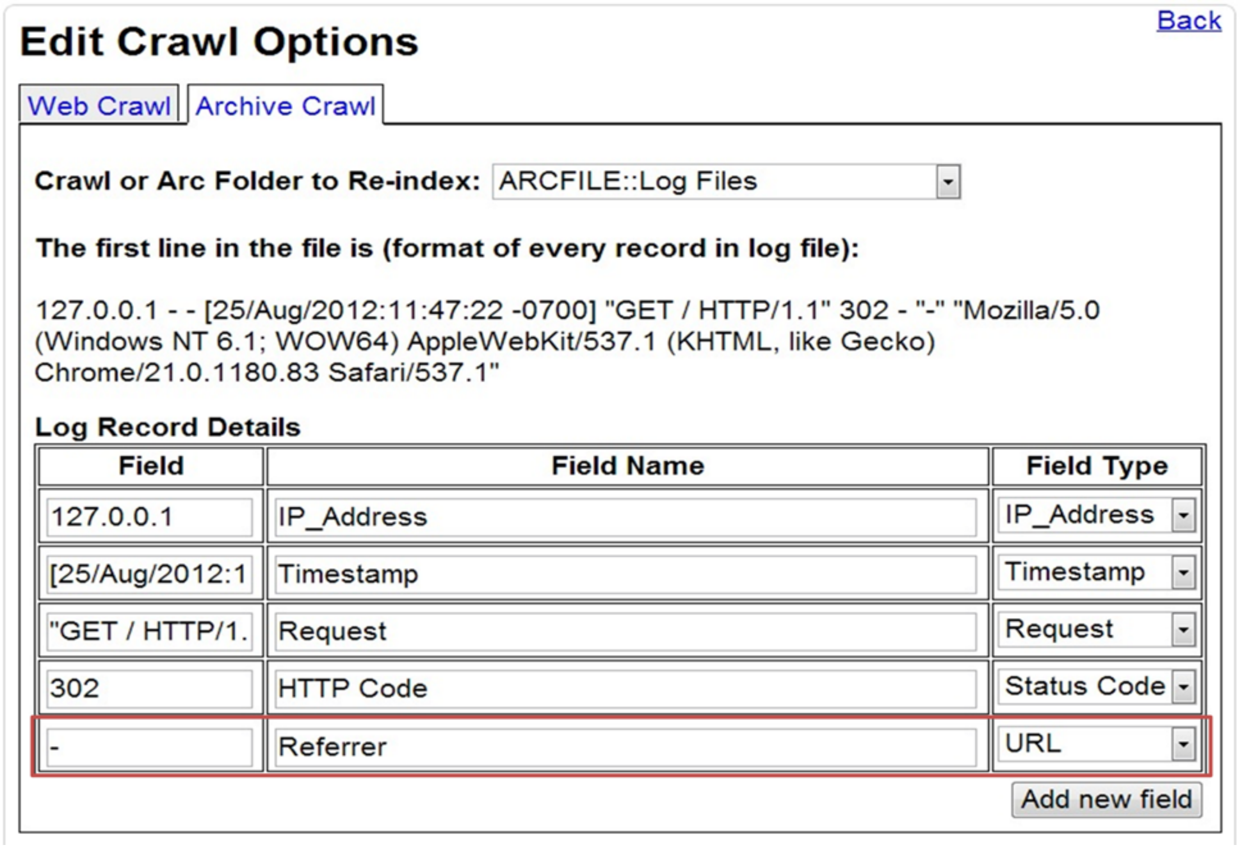
Crawl or Arc Folder to Re-index: ARCFILE::Log Files

The first line in the file is (format of every record in log file):

127.0.0.1 - - [25/Aug/2012:11:47:22 -0700] "GET / HTTP/1.1" 302 - "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML, like Gecko) Chrome/21.0.1180.83 Safari/537.1"

Log Record Details

Field	Field Name	Field Type
127.0.0.1	IP_Address	IP_Address
		IP_Address



Back

Edit Crawl Options

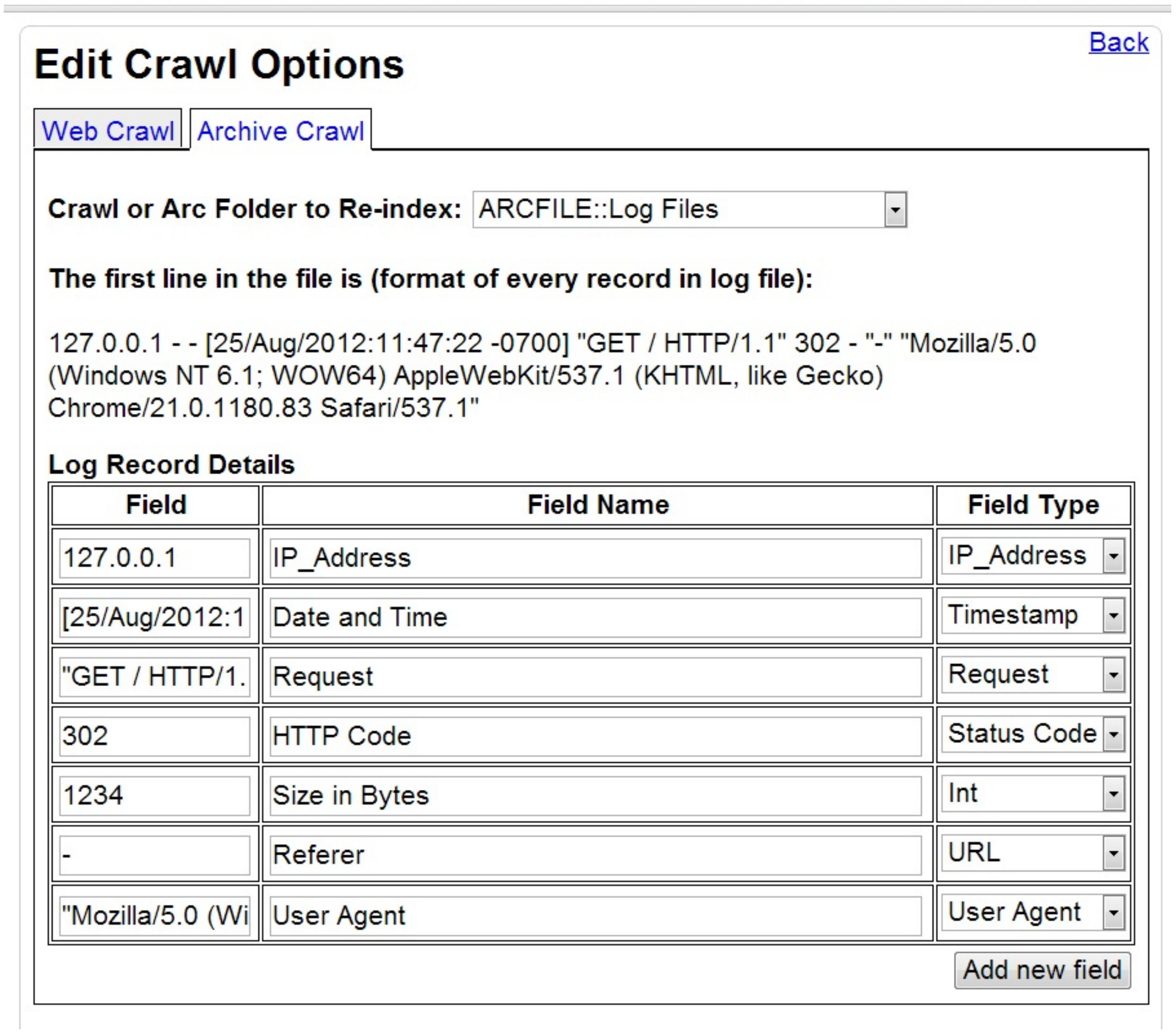
Crawl or Arc Folder to Re-index: ARCFILE::Log Files

The first line in the file is (format of every record in log file):

127.0.0.1 - - [25/Aug/2012:11:47:22 -0700] "GET / HTTP/1.1" 302 - "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML, like Gecko) Chrome/21.0.1180.83 Safari/537.1"

Log Record Details

Field	Field Name	Field Type
127.0.0.1	IP_Address	IP_Address
[25/Aug/2012:1	Timestamp	Timestamp
"GET / HTTP/1.	Request	Request
302	HTTP Code	Status Code
-	Referrer	URL



Edit Crawl Options

Crawl or Arc Folder to Re-index: ARCFILE::Log Files

The first line in the file is (format of every record in log file):

```
127.0.0.1 - - [25/Aug/2012:11:47:22 -0700] "GET / HTTP/1.1" 302 - "-" "Mozilla/5.0
(Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML, like Gecko)
Chrome/21.0.1180.83 Safari/537.1"
```

Log Record Details

Field	Field Name	Field Type
127.0.0.1	IP_Address	IP_Address
[25/Aug/2012:1	Date and Time	Timestamp
"GET / HTTP/1.	Request	Request
302	HTTP Code	Status Code
1234	Size in Bytes	Int
-	Referer	URL
"Mozilla/5.0 (Wi	User Agent	User Agent

Figure 10: Steps to enter details for crawling the log files

3.1.2 Database Records

The user interface for database records is limited to the database connection details. A simple user interface which asks the user for the database connection details is displayed as soon as the user selects the databases option from the re-crawls list. The database connection details include the hostname, username, password, database name and the database query.

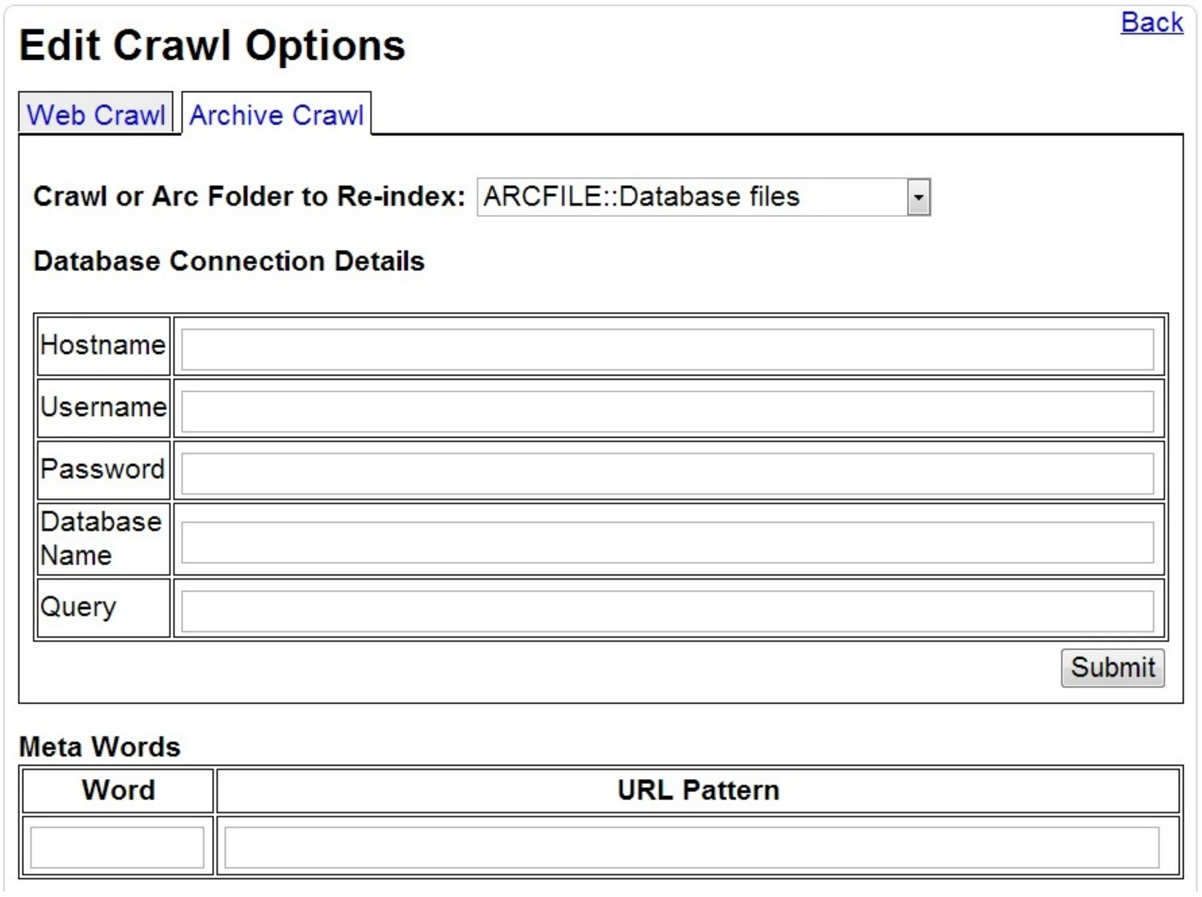
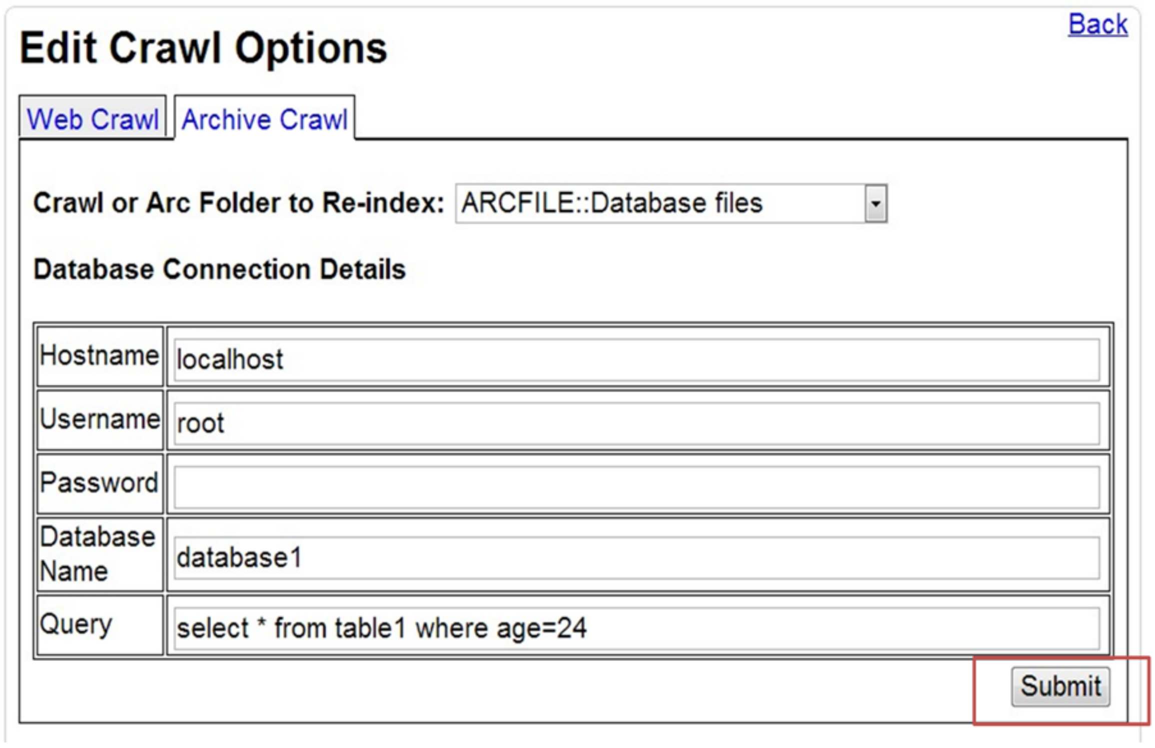


Figure 11: Databases Interface


Edit Crawl Options

Crawl or Arc Folder to Re-index: ARCFILE::Database files

Database Connection Details

Hostname	localhost
Username	root
Password	
Database Name	database1
Query	select * from table1 where age=24

Figure 12: Database connection details entered

After all the database connection details are entered, the submit button should be clicked and all the details are saved. The host name, user name and password are used to connect to the host whereas the database name is used to establish a connection to the database. The query specified by the user is used to limit the records to crawl and is used to retrieve the records from the database. The records returned based on the query are only crawled and indexed and not all the database records in the table.

3.2 Backend Design

The backend design includes the process design that is followed to implement the features in Yioop. The features added to Yioop include the crawling and indexing of log files, databases and handling the short links correctly in Yioop. The modifications to Yioop architecture and the process flow for each feature are described in this section.

3.2.1 URL Shortening

In order to get Yioop to work with URL shortened links, it was required to make a study on how Yioop handled theses links previously. The fetcher.php file is responsible for handling all the pages and crawling them. This file had an array named "site" in the fetcher.php file which stores all the attributes and information pertaining to a particular page being crawled and indexed by Yioop. The attributes include location of the page, links, HTTP code, title etc. After the crawl on bitly link, it was observed that all the bit.ly links possessed an HTTP status code greater than 300. A 301 redirect to the original link was being done and the original link was stored in the location attribute of the URL. The URL attribute of site stores the actual URL which is ranked. Yioop handled the short URLs in the same way it handled any other links. Therefore, modifications needed to be done for this purpose.

Therefore, the code to be developed for this purpose required changes in some of the existing files of Yioop. There were no new files that were added to the Yioop architecture. The implementation section would describe in detail the changes done to the files in Yioop to achieve the feature.

3.2.2 Log Files

In order to write an archive iterator for Yioop to index log files, all the iterators currently available were studied. An archive bundle iterator in Yioop has the following process design.

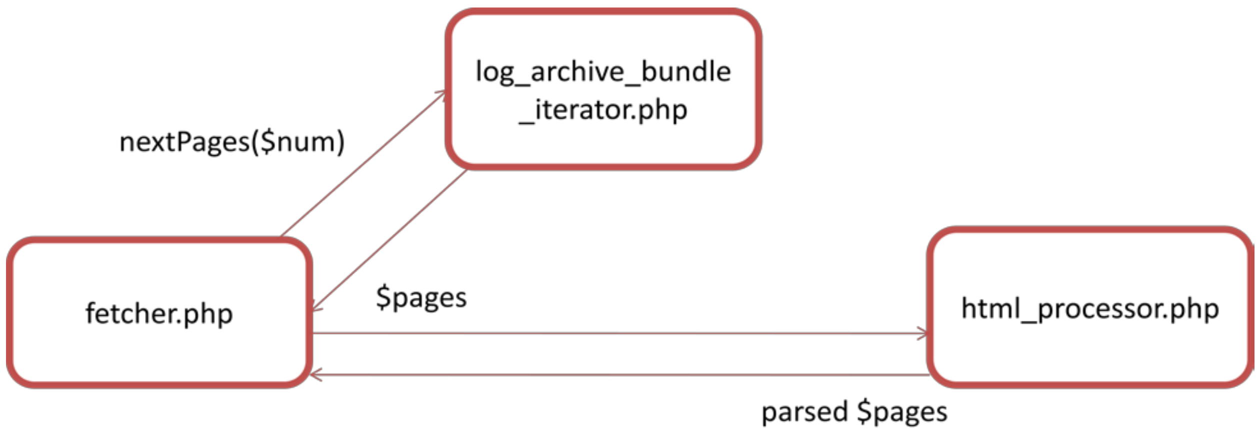


Figure 13: Process design for archive iterator in Yioop

In the log_archive_bundle_iterator.php, the above process design is followed accordingly and necessary functionalities were achieved. A function is written in order to parse the log records and the results are sent to the nextPages() function which are in turn converted into html pages in nextPage() function.

The design flow of the crawling of log files is shown below. The numbers in the figure indicate the sequence of steps that occur in the crawling.

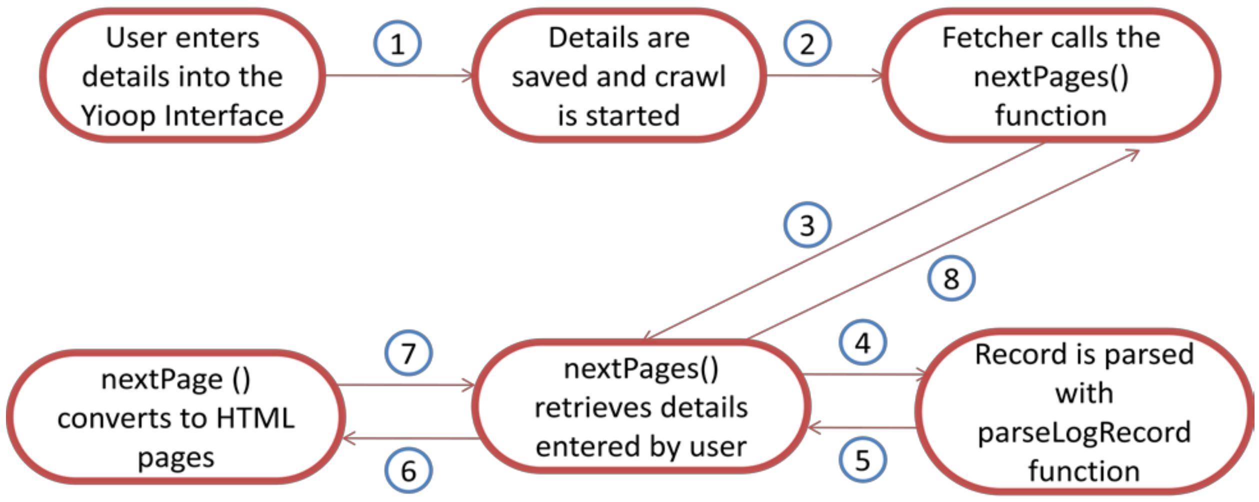

User enters details into the Yioop Interface
1
Details are saved and crawl is started
2
Fetcher calls the nextPages() function
3
8
nextPage () converts to HTML pages
7
6
nextPages() retrieves details entered by user
4
5
Record is parsed with parseLogRecord function

Figure 14: Sequence of steps in the design flow of crawling and indexing log files

3.2.3 Database Records

The process of crawling database records is similar to that of crawling log records with minor differences. The crawling of database records require the user to enter the database connection details and also allows the user to limit the records to crawl based on the crawl. The design flow is shown in the below figure.

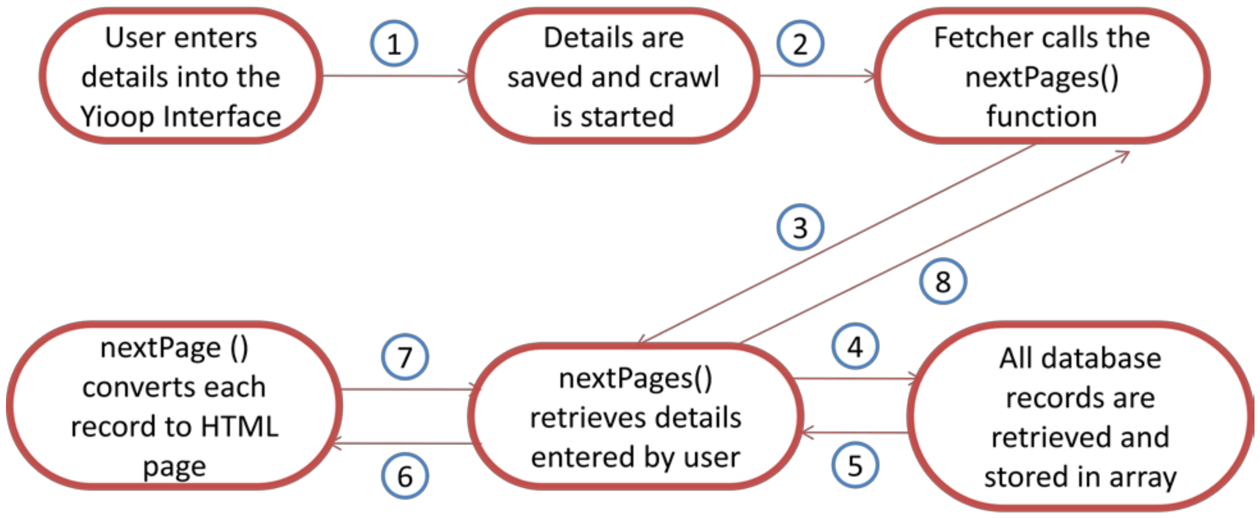

User enters details into the Yioop Interface
1
Details are saved and crawl is started
2
Fetcher calls the nextPages() function
3
8
nextPage () converts each record to HTML page
7
6
nextPages() retrieves details entered by user
4
5
All database records are retrieved and stored in array

Figure 15: Sequence of steps in the design flow of crawling database records

4 Implementation

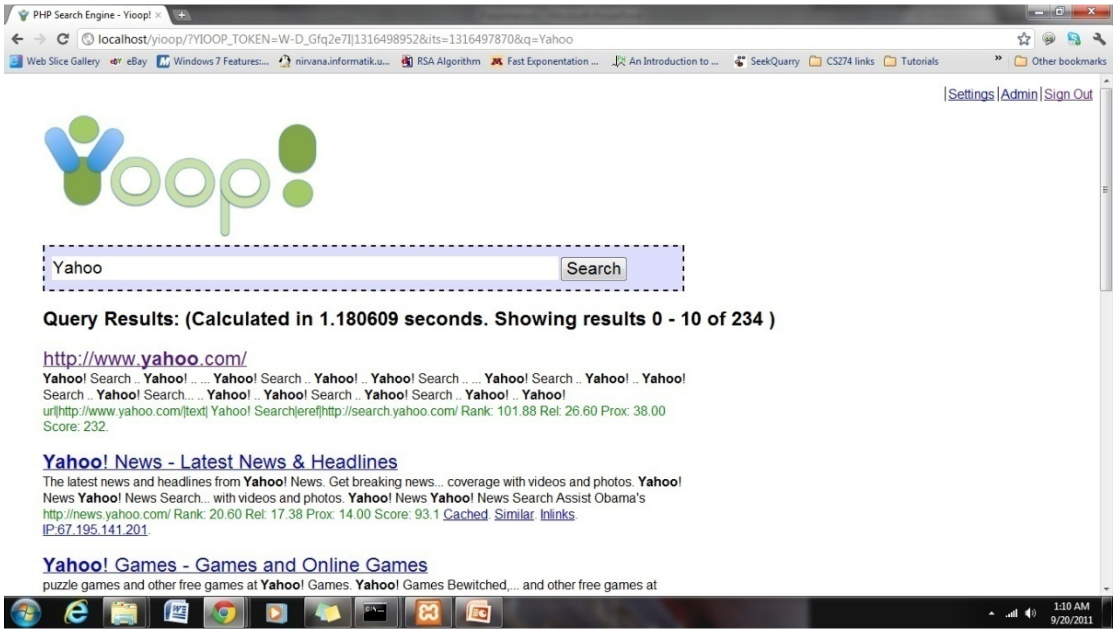
This section describes how each of the features described in the above sections have been implemented in Yioop. The content in this section provides a description of the changes done to the existing files in Yioop, description of programs that are newly added to Yioop to accomplish the features and also the details of the implementation of user interface described in the user interface design subsection of the design section. This implementation section mainly describes the changes done in Yioop files in terms of code. This section also contains code snippets.

4.1 URL Shortening Links

In order to get Yioop running correctly with the short links, it was necessary to know how Yioop deals with these links previously. In order to perform this task, crawls were performed in Yioop with different we crawl options specified before starting the crawl. The first crawl was performed with Yioop's default options and domains whereas the second crawl was performed only on a single bit.ly link. The results obtained for the keyword Yahoo when a bit.ly link of Yahoo was crawled are shown below.

Yioop has an array named "site" in the fetcher.php file which stores all the attributes and information pertaining to a particular page being crawled and indexed by Yioop. The attributes include location of the page, links, HTTP code, title etc. After the crawl on bitly link, it was observed that all the bit.ly links possessed an HTTP status code greater than 300. A 301 redirect to the original link was being done and the original link was stored in the location attribute of the URL. The URL attribute of site stores the actual URL which is ranked. Yioop handled the short URLs in the same way it handled any other links.

In the code added to this project, Yioop was modified such that it treats the URLs with response code greater than 300 in a different way. When Yioop encounters a URL with HTTP code greater than 300, code was written to retrieve the location of the link which gives the original URL of the short URL. The URL attribute of the site array is pointed to the original link so that the rank is assigned to the original link. This way, the short URLs are made to be handled correctly by Yioop.


PHP Search Engine - Yioop!
localhost/yioop/?YIOOP_TOKEN=W-D_Gfq2e7I|1316498952&its=1316497870&q=Yahoo
|Settings|Admin|Sign Out
Yioop!
Search

Yahoo

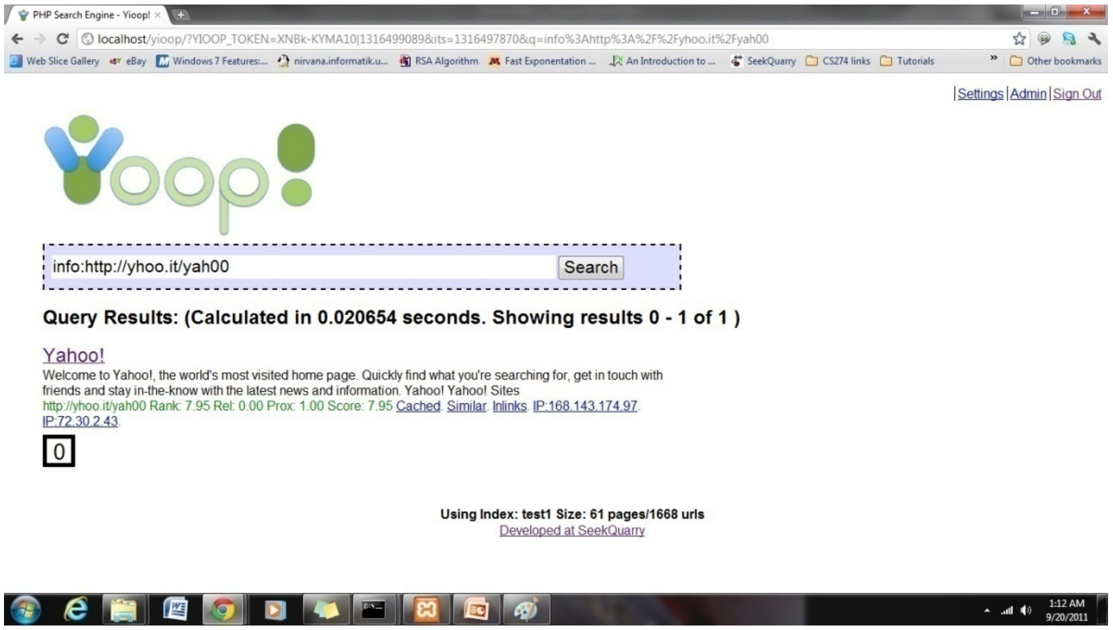
Query Results: (Calculated in 1.180609 seconds. Showing results 0 - 10 of 234)

http://www.**yahoo**.com/
Yahoo! Search .. **Yahoo**! .. ... **Yahoo**! Search .. **Yahoo**! .. **Yahoo**! Search .. ... **Yahoo**! Search .. **Yahoo**! .. **Yahoo**! Search .. **Yahoo**! Search... .. **Yahoo**! .. **Yahoo**! Search .. **Yahoo**! Search .. **Yahoo**! .. **Yahoo**!
url|http://www.yahoo.com/|text| Yahoo! Search|eref|http://search.yahoo.com/ Rank: 101.88 Rel: 26.60 Prox: 38.00 Score: 232.

Yahoo! News - Latest News & Headlines
The latest news and headlines from **Yahoo**! News. Get breaking news... coverage with videos and photos. **Yahoo**! News **Yahoo**! News Search... with videos and photos. **Yahoo**! News **Yahoo**! News Search Assist Obama's
http://news.yahoo.com/ Rank: 20.60 Rel: 17.38 Prox: 14.00 Score: 93.1 Cached. Similar. Inlinks. IP:67.195.141.201.

Yahoo! Games - Games and Online Games
puzzle games and other free games at **Yahoo**! Games. **Yahoo**! Games Bewitched,... and other free games at

Figure 16: Search results for Yahoo after the crawl


PHP Search Engine - Yioop!
localhost/yioop/?YIOOP_TOKEN=XNBk-KYMA10|1316499089&its=1316497870&q=info%3Ahttp%3A%2F%2Fyhoo.it%2Fyah00
|Settings|Admin|Sign Out
Yioop!
Search

info:http://yhoo.it/yah00

Query Results: (Calculated in 0.020654 seconds. Showing results 0 - 1 of 1)

Yahoo!
Welcome to Yahoo!, the world's most visited home page. Quickly find what you're searching for, get in touch with friends and stay in-the-know with the latest news and information. Yahoo! Yahoo! Sites
http://yhoo.it/yah00 Rank: 7.95 Rel: 0.00 Prox: 1.00 Score: 7.95 Cached. Similar. Inlinks. IP:168.143.174.97. IP:72.30.2.43.

0

Using Index: test1 Size: 61 pages/1668 urls
Developed at SeekQuarry

Figure 17: 301 redirect to the Yahoo link

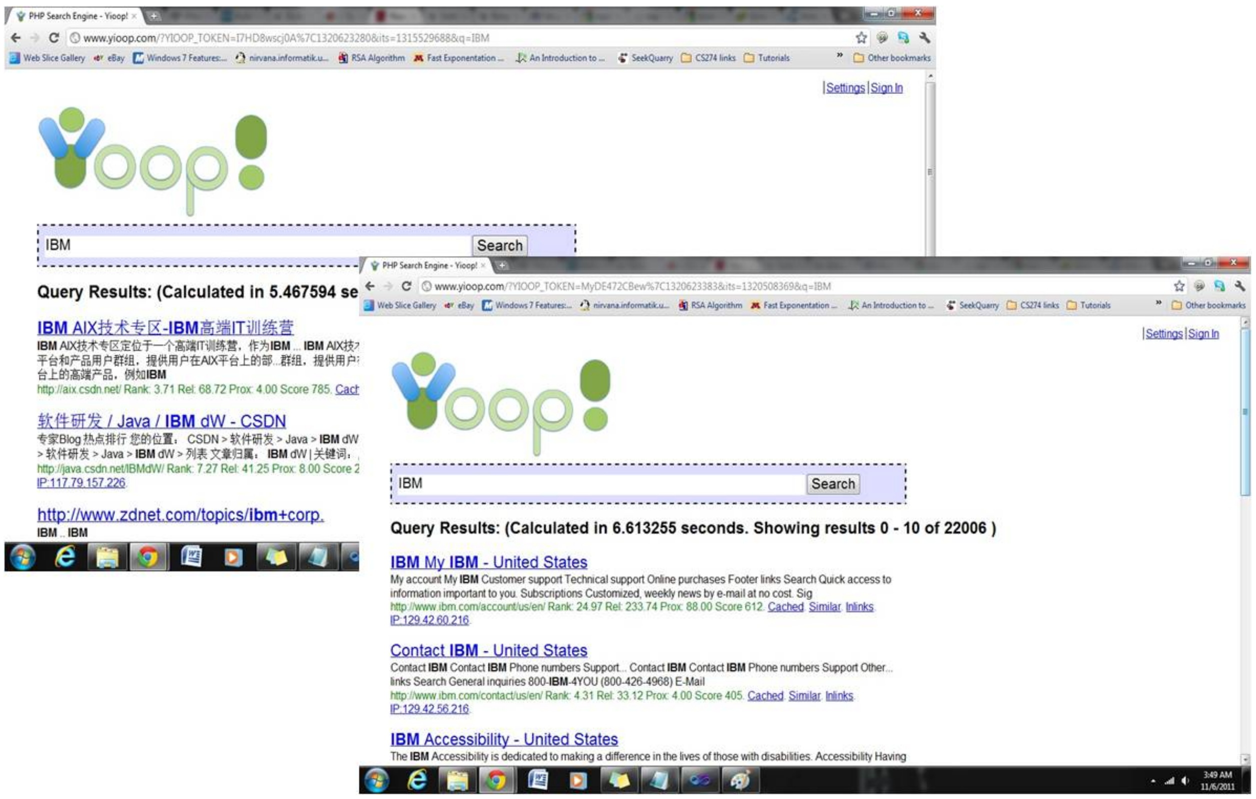
In order to make Yioop work correctly with these links, the code was added in three files of Yioop.

crawl_constants.php - A new constant has been added in crawl_constants.php file for storing the Location" attribute which has the original link where the short link service points

fetch_url.php **-** The extraction of the "Location" information from the short URL is done in this file. This information was required to point the results to the original URL and not the short URL.

fetcher.php - The final addition of code was done in this file. This step changes the short URL to the original link using the location information extracted from fetch_url.php. The information retrieved is assigned to the URL attribute of the site array in order to point to the original links.

After the addition of code into all these files, the short URL is changed to the original link and the crawling was performed on the original link and the search results displayed reflected the same. The search results can be seen in the following figures.



Query Results: (Calculated in 5.467594 se

IBM AIX技术专区-IBM高端IT训练营
IBM AIX技术专区定位于一个高端IT训练营，作为IBM ... IBM AIX技
平台和产品用户群组，提供用户在AIX平台上的部...群组，提供用户
台上的高端产品，例如IBM
http://aix.csdn.net/ Rank: 3.71 Rel: 68.72 Prox: 4.00 Score 785. Cach

软件研发 / Java / IBM dW - CSDN
专家Blog 热点排行 您的位置： CSDN > 软件研发 > Java > IBM dW
> 软件研发 > Java > IBM dW > 列表 文章归属： IBM dW | 关键词：
http://java.csdn.net/IBMdW/ Rank: 7.27 Rel: 41.25 Prox: 8.00 Score 2
IP:117.79.157.226.

http://www.zdnet.com/topics/ibm+corp.
IBM .. IBM

PHP Search Engine - Yioop!
www.yioop.com/?YIOOP_TOKEN=MyDE472CBew%7C1320623383&its=1320508369&q=IBM
|Settings|Sign In
IBM
Search
3:49 AM
11/6/2011

Query Results: (Calculated in 6.613255 seconds. Showing results 0 - 10 of 22006)

IBM My IBM - United States
My account My IBM Customer support Technical support Online purchases Footer links Search Quick access to information important to you. Subscriptions Customized, weekly news by e-mail at no cost. Sig
http://www.ibm.com/account/us/en/ Rank: 24.97 Rel: 233.74 Prox: 88.00 Score 612. Cached. Similar. Inlinks.
IP:129.42.60.216.

Contact IBM - United States
Contact IBM Contact IBM Phone numbers Support... Contact IBM Contact IBM Phone numbers Support Other... links Search General inquiries 800-IBM-4YOU (800-426-4968) E-Mail
http://www.ibm.com/contact/us/en/ Rank: 4.31 Rel: 33.12 Prox: 4.00 Score 405. Cached. Similar. Inlinks.
IP:129.42.56.216.

IBM Accessibility - United States
The IBM Accessibility is dedicated to making a difference in the lives of those with disabilities. Accessibility Having

Figure 18: Results shown before and after the code was added

The figures show that before the code was added to Yioop, the results consisted of unimportant links and not pertaining to the actual IBM domain whereas after the code was added, the search results consisted of pages which are of IBM domain and also some important links of IBM.

4.2 Indexing Log Files in Yioop!

In order to add indexing of log files in Yioop, two tasks needed to be performed:

(1) Build a user interface for the log files

(2) Create an archive bundle iterator

The user interface for log files needed to be created in the archive crawl options section of the edit crawl options interface in Yioop. This interface is shown to the user only if he chooses to crawl the log files option from the drop down provided in the archive crawl options interface. The user interface provides user the options to enter the log fields, field names for them and also the field type. The user is also given the option to add any number of fields as per the first sample record displayed to the user by clicking on add new field button. Once the user clicks on save options after entering all the required fields, the options for the particular crawl are saved. In order to implement this user interface, code was added in the following files of Yioop.

admin_controller.php - A new array for log records has been created in this file in order to store the details of the log records entered by the user in the web interface of Yioop.

crawloptions_element.php - The UI that needs to be displayed when log files is selected as an option is created in this file. Also, the code is added in this file to show the user interface only when log files is selected from the drop down menu of the recrawl options. Also, it retrieves the details of inputs given by the user from the array stored in the admin_controller.php program and displays the options. It also has the code to save the options entered by the user in a file into the folder created by the user to store his log files. The table shown in the interface, "Add new field" button in the interface are also added in this file.

search.css- The code is added here to specify the styles for the log records table shown in the web interface to the user. The borders, width, padding, alignment etc are set in this file.

After the addition of code in all the three files of Yioop, the user interface for log files is created and is shown on Yioop when the user selects the log files option from archive tab options from the edit crawl options.

In order to crawl and index the log records in Yioop, an archive bundle iterator was added into the yioop/lib/archive_bundle_iterators folder. The file that is added into the folder was log_archive_bundle_iterator.php.

The log_archive_bundle_iterator.php file consisted of five major functions:

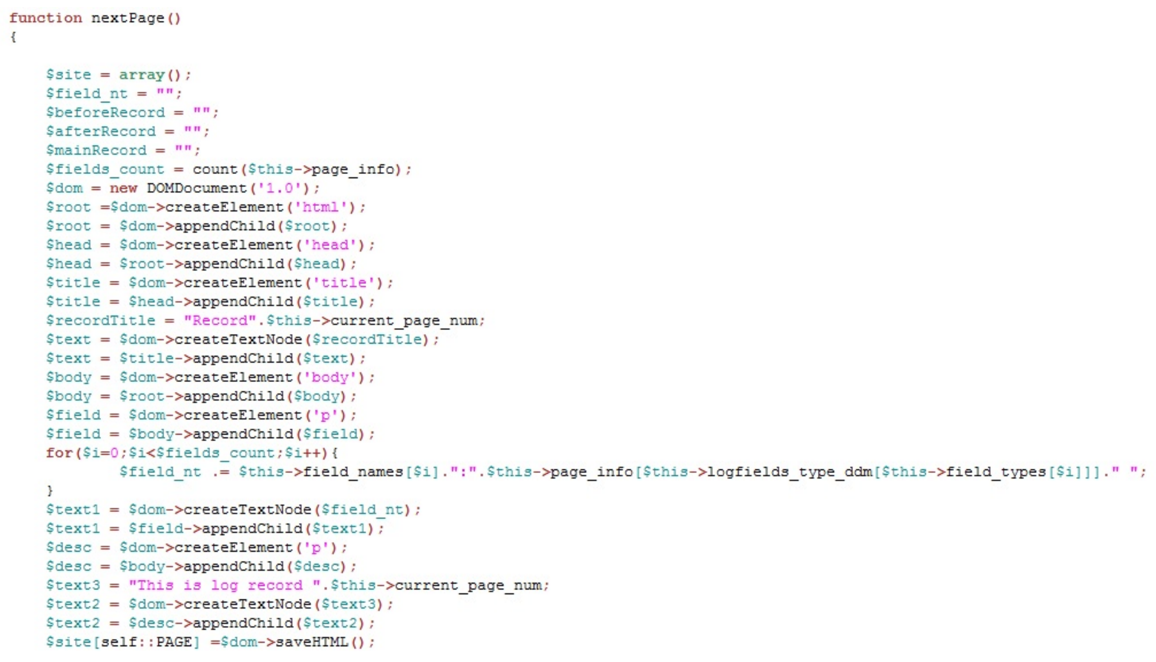
__construct - This is the constructor responsible for creating an instance of an archive iterator with timestamps and also for calling the createMasterAndRecords function.

createMasterandRecords - This would create the master.log file which would contain all the log records from all the log files included in the folder created by the user. All the log records are put into one file in order to make the processing easy.

parseLogRecord - This function is called for every log record and it parses the record according to the regular expressions and returns the results into a global array.

nextPages - The nextPages function would have a loop running which calls the nextPage function to convert each of the records into a web page, i.e., a HTML page to be indexed by Yioop.

nextPage - The nextPage function is responsible for creating the HTML pages of each log record. Changes made in nextPage function will change the web page of the log record or the database record. The title of the web page can be changed in this nextPage function. The URL of the page can be changed in the nextPage function and can be pointed to any file you want to show to the user along with the details of the log record. One can also add new text nodes to the web page in order to mention anything interesting about the log record. Therefore, nextPage becomes an important function in the archive bundle iterators.



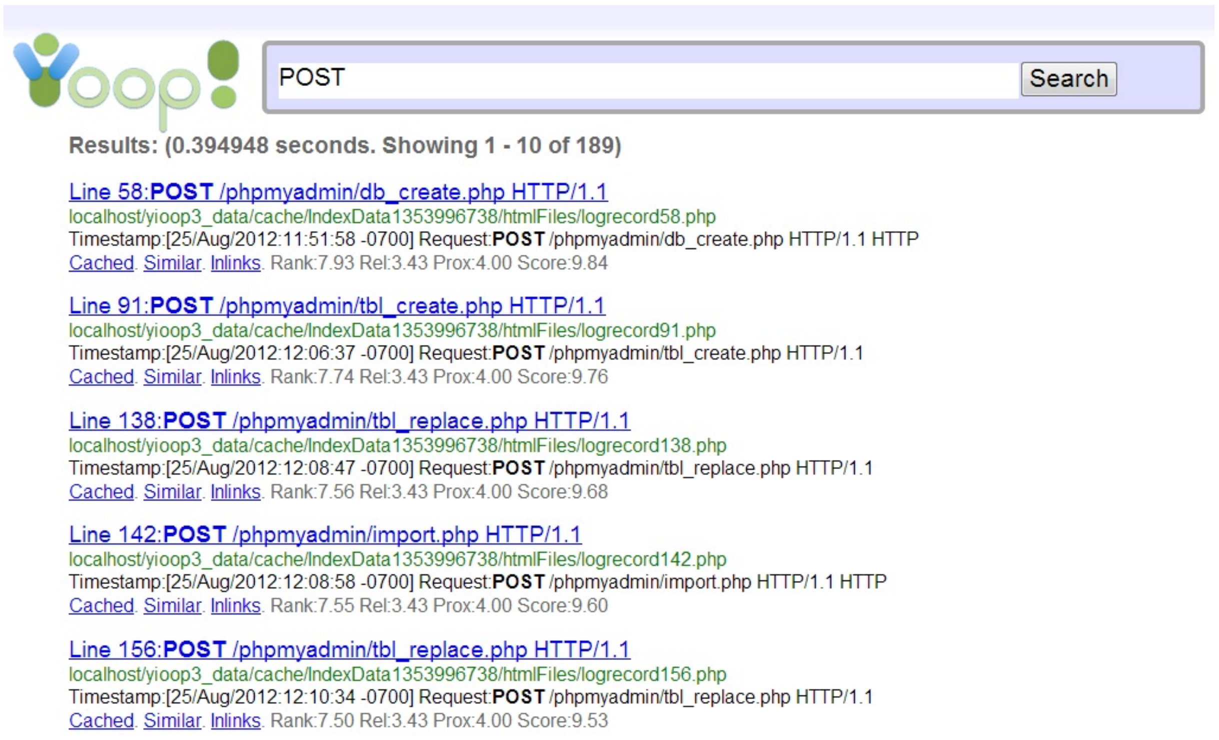
```
function nextPage()
{

    $site = array();
    $field_nt = "";
    $beforeRecord = "";
    $afterRecord = "";
    $mainRecord = "";
    $fields_count = count($this->page_info);
    $dom = new DOMDocument('1.0');
    $root =$dom->createElement('html');
    $root = $dom->appendChild($root);
    $head = $dom->createElement('head');
    $head = $root->appendChild($head);
    $title = $dom->createElement('title');
    $title = $head->appendChild($title);
    $recordTitle = "Record".$this->current_page_num;
    $text = $dom->createTextNode($recordTitle);
    $text = $title->appendChild($text);
    $body = $dom->createElement('body');
    $body = $root->appendChild($body);
    $field = $dom->createElement('p');
    $field = $body->appendChild($field);
    for($i=0;$i<$fields_count;$i++){
            $field_nt .= $this->field_names[$i].":".$this->page_info[$this->logfields_type_ddm[$this->field_types[$i]]]." ";
    }
    $text1 = $dom->createTextNode($field_nt);
    $text1 = $field->appendChild($text1);
    $desc = $dom->createElement('p');
    $desc = $body->appendChild($desc);
    $text3 = "This is log record ".$this->current_page_num;
    $text2 = $dom->createTextNode($text3);
    $text2 = $desc->appendChild($text2);
    $site[self::PAGE] =$dom->saveHTML();
```

Figure 19: nextPage function for creating HTML pages

Apart from the functions mentioned above, there are also other functions such as saveCheckpoint, restoreCheckpoint, getFieldDetails, reset, weight etc, which perform the necessary functions required in the archive iterator.

After the implementation of archive bundle iterator, a crawl was performed with log records with the options shown in Figure 10. A crawl was started and is set as index after the crawl was stopped the following are the results obtained after the crawl. Also, each record in the log files is named after the request field in the log record. In case a log record does not contain any request information the records are by default named as "Line(Number of the record-1)". For example, the 601th record is named as "Line600" as the count starts from 0. This number indicates the line number of a particular log record in a log file.



Results: (0.394948 seconds. Showing 1 - 10 of 189)

Line 58:**POST** /phpmyadmin/db_create.php HTTP/1.1
localhost/yioop3_data/cache/IndexData1353996738/htmlFiles/logrecord58.php
Timestamp:[25/Aug/2012:11:51:58 -0700] Request:**POST** /phpmyadmin/db_create.php HTTP/1.1 HTTP
Cached. Similar. Inlinks. Rank:7.93 Rel:3.43 Prox:4.00 Score:9.84

Line 91:**POST** /phpmyadmin/tbl_create.php HTTP/1.1
localhost/yioop3_data/cache/IndexData1353996738/htmlFiles/logrecord91.php
Timestamp:[25/Aug/2012:12:06:37 -0700] Request:**POST** /phpmyadmin/tbl_create.php HTTP/1.1
Cached. Similar. Inlinks. Rank:7.74 Rel:3.43 Prox:4.00 Score:9.76

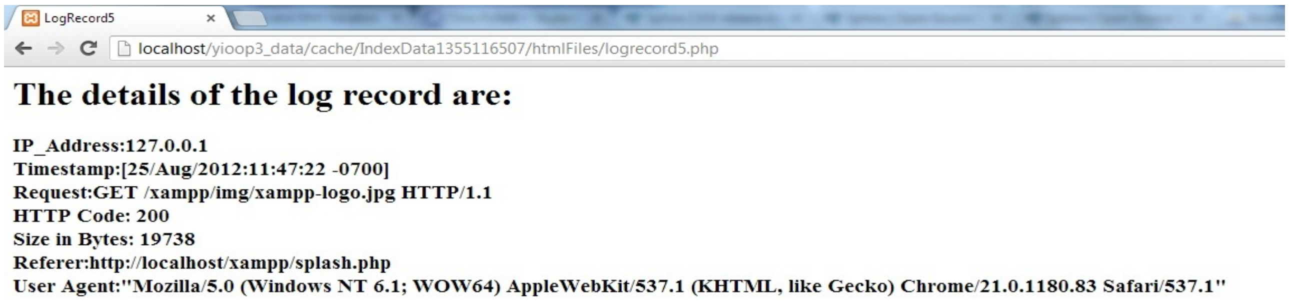
Line 138:**POST** /phpmyadmin/tbl_replace.php HTTP/1.1
localhost/yioop3_data/cache/IndexData1353996738/htmlFiles/logrecord138.php
Timestamp:[25/Aug/2012:12:08:47 -0700] Request:**POST** /phpmyadmin/tbl_replace.php HTTP/1.1
Cached. Similar. Inlinks. Rank:7.56 Rel:3.43 Prox:4.00 Score:9.68

Line 142:**POST** /phpmyadmin/import.php HTTP/1.1
localhost/yioop3_data/cache/IndexData1353996738/htmlFiles/logrecord142.php
Timestamp:[25/Aug/2012:12:08:58 -0700] Request:**POST** /phpmyadmin/import.php HTTP/1.1 HTTP
Cached. Similar. Inlinks. Rank:7.55 Rel:3.43 Prox:4.00 Score:9.60

Line 156:**POST** /phpmyadmin/tbl_replace.php HTTP/1.1
localhost/yioop3_data/cache/IndexData1353996738/htmlFiles/logrecord156.php
Timestamp:[25/Aug/2012:12:10:34 -0700] Request:**POST** /phpmyadmin/tbl_replace.php HTTP/1.1
Cached. Similar. Inlinks. Rank:7.50 Rel:3.43 Prox:4.00 Score:9.53

Figure 20: Search results obtained after crawling log records

Each of the records when opened show a web page which shows the record details in detail. Each field extracted and the field name given for each of them by the user are displayed in this page. Also, one can search for a particular query and the search results return the records which consist of that search query. The two figures below show the search results in Yioop after crawling and indexing the log records.

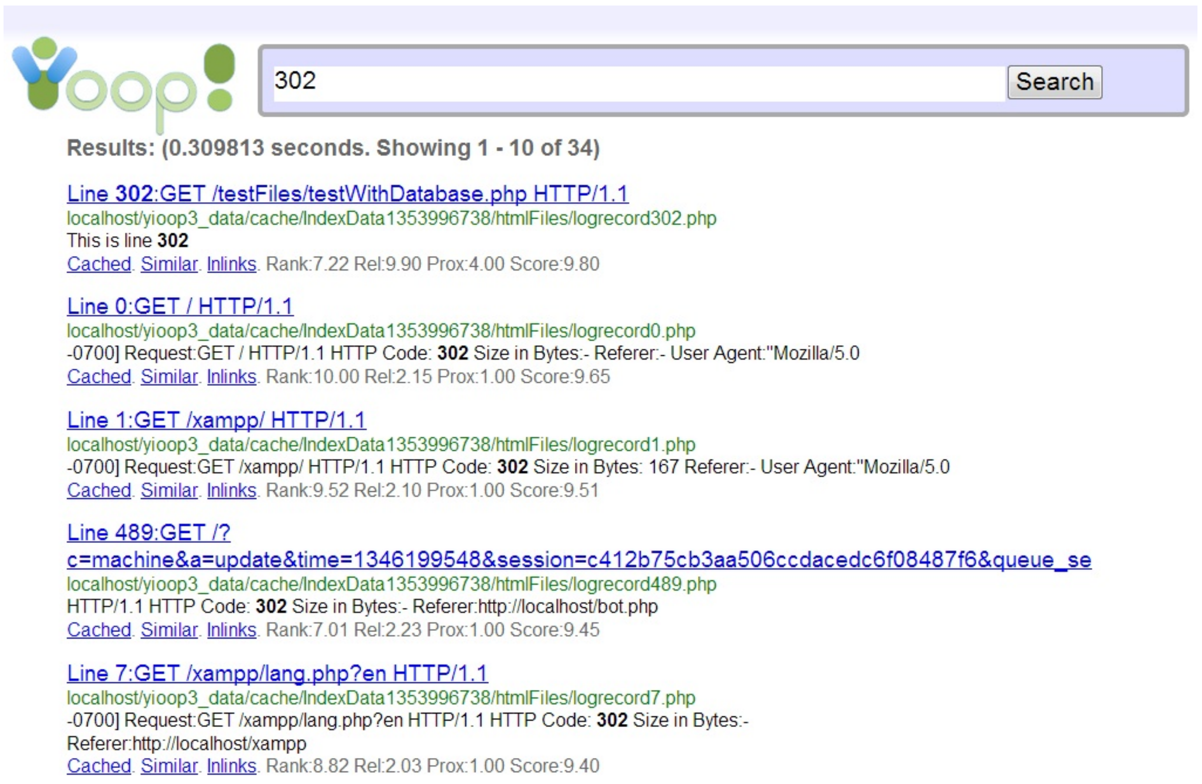

LogRecord5

localhost/yioop3_data/cache/IndexData1355116507/htmlFiles/logrecord5.php

The details of the log record are:

IP_Address:127.0.0.1
Timestamp:[25/Aug/2012:11:47:22 -0700]
Request:GET /xampp/img/xampp-logo.jpg HTTP/1.1
HTTP Code: 200
Size in Bytes: 19738
Referer:http://localhost/xampp/splash.php
User Agent:"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML, like Gecko) Chrome/21.0.1180.83 Safari/537.1"

Figure 21: An open log record


302 Search

Results: (0.309813 seconds. Showing 1 - 10 of 34)

Line **302**:GET /testFiles/testWithDatabase.php HTTP/1.1
localhost/yioop3_data/cache/IndexData1353996738/htmlFiles/logrecord302.php
This is line **302**
Cached. Similar. Inlinks. Rank:7.22 Rel:9.90 Prox:4.00 Score:9.80

Line 0:GET / HTTP/1.1
localhost/yioop3_data/cache/IndexData1353996738/htmlFiles/logrecord0.php
-0700] Request:GET / HTTP/1.1 HTTP Code: **302** Size in Bytes:- Referer:- User Agent:"Mozilla/5.0
Cached. Similar. Inlinks. Rank:10.00 Rel:2.15 Prox:1.00 Score:9.65

Line 1:GET /xampp/ HTTP/1.1
localhost/yioop3_data/cache/IndexData1353996738/htmlFiles/logrecord1.php
-0700] Request:GET /xampp/ HTTP/1.1 HTTP Code: **302** Size in Bytes: 167 Referer:- User Agent:"Mozilla/5.0
Cached. Similar. Inlinks. Rank:9.52 Rel:2.10 Prox:1.00 Score:9.51

Line 489:GET /?c=machine&a=update&time=1346199548&session=c412b75cb3aa506ccdacedc6f08487f6&queue_se
localhost/yioop3_data/cache/IndexData1353996738/htmlFiles/logrecord489.php
HTTP/1.1 HTTP Code: **302** Size in Bytes:- Referer:http://localhost/bot.php
Cached. Similar. Inlinks. Rank:7.01 Rel:2.23 Prox:1.00 Score:9.45

Line 7:GET /xampp/lang.php?en HTTP/1.1
localhost/yioop3_data/cache/IndexData1353996738/htmlFiles/logrecord7.php
-0700] Request:GET /xampp/lang.php?en HTTP/1.1 HTTP Code: **302** Size in Bytes:-
Referer:http://localhost/xampp
Cached. Similar. Inlinks. Rank:8.82 Rel:2.03 Prox:1.00 Score:9.40

Figure 22: Search for all log records with a 302 status code

4.3 Indexing Database Records in Yioop!

In order to add indexing of database files in Yioop, there were two tasks performed:

(1) Building a user interface for the database files

(2) Creating an archive bundle iterator for database files

The user interface was implemented by adding code into the following files of Yioop. It needs to be seen in the same section as log files was shown. It is displayed when database files is chosen as the option to crawl.

admin_controller.php - A new array has been created in this file in order to store the database connection details of the log records entered by the user in the web interface of Yioop.

crawloptions_element.php - The UI that needs to be displayed when databases is selected as an option is created in this file. The table displayed to the user to enter the details is built in this file.

search.css- The code is added here to specify the styles for the database connections table shown in the web interface to the user.

After the addition of code in all the three files of Yioop, the user interface for database connection table is created and is shown on Yioop when the user selects the databases option from archive tab options from the edit crawl options. The code looks similar to that log records and therefore is not shown here.

In order to crawl and index the database records in Yioop, an archive bundle iterator was added into the yioop/lib/archive_bundle_iterators folder. The file that is added into the folder was database_archive_bundle_iterator.php. This file consisted of three major functions:

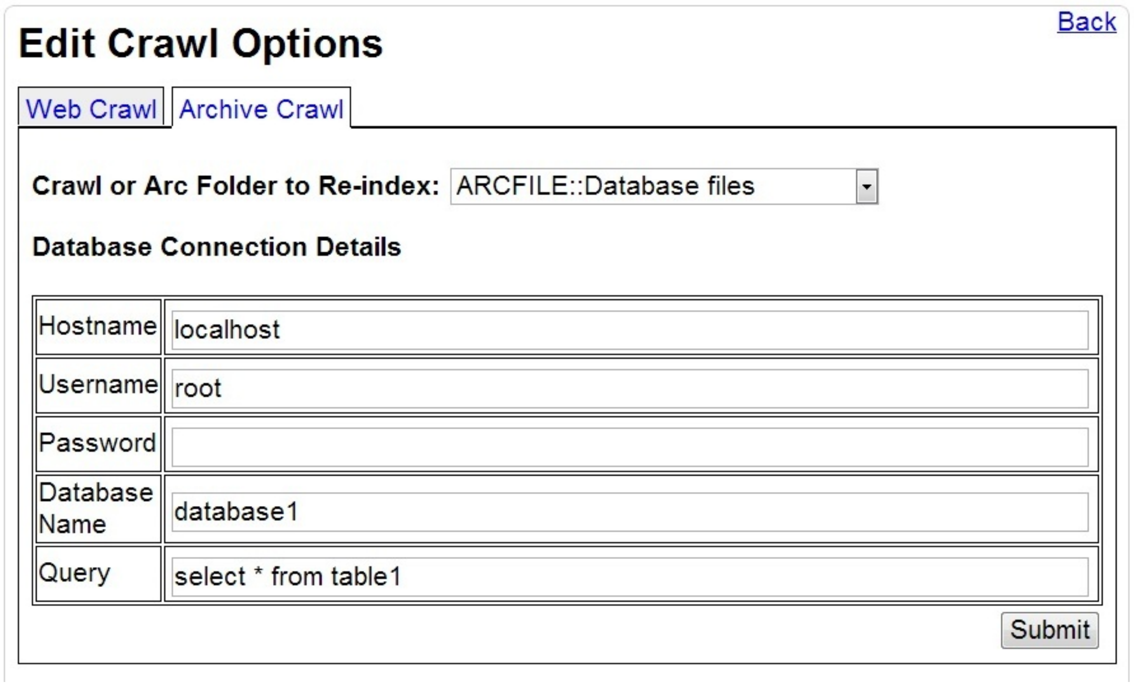
createRecords - This would create the array of records after executing the query specified by the user. This array is later used to process the records.

nextPages - The nextPages function would have a loop running which calls the nextPage function to convert each of the database records into a web page, i.e., a HTML page to be indexed by Yioop.

nextPage - The nextPage function is responsible for creating the HTML page with the database record.

Apart from the functions mentioned above, there are also other functions such as saveCheckpoint, restoreCheckpoint, reset, weight, a constructor etc, which perform the necessary functions required in the database archive iterator.

After the implementation of archive bundle iterator, a crawl was performed with database records with the options shown in Figure 23. A crawl was started and is set as index after the crawl was stopped the following are the results obtained after the crawl. Also, each record in the database is named as "DatabaseRecord(Number of the record-1)". For example, the 101th record is named as "Database Record100" as the count starts from 0. The record when opened will display the details of the particular record to the user. The search on a query would return the records which have that search query in them. A search on the keyword Srujana is done on the crawl of database records. There was one record with that name and the record is returned. The results are shown in Figure 24.



Back

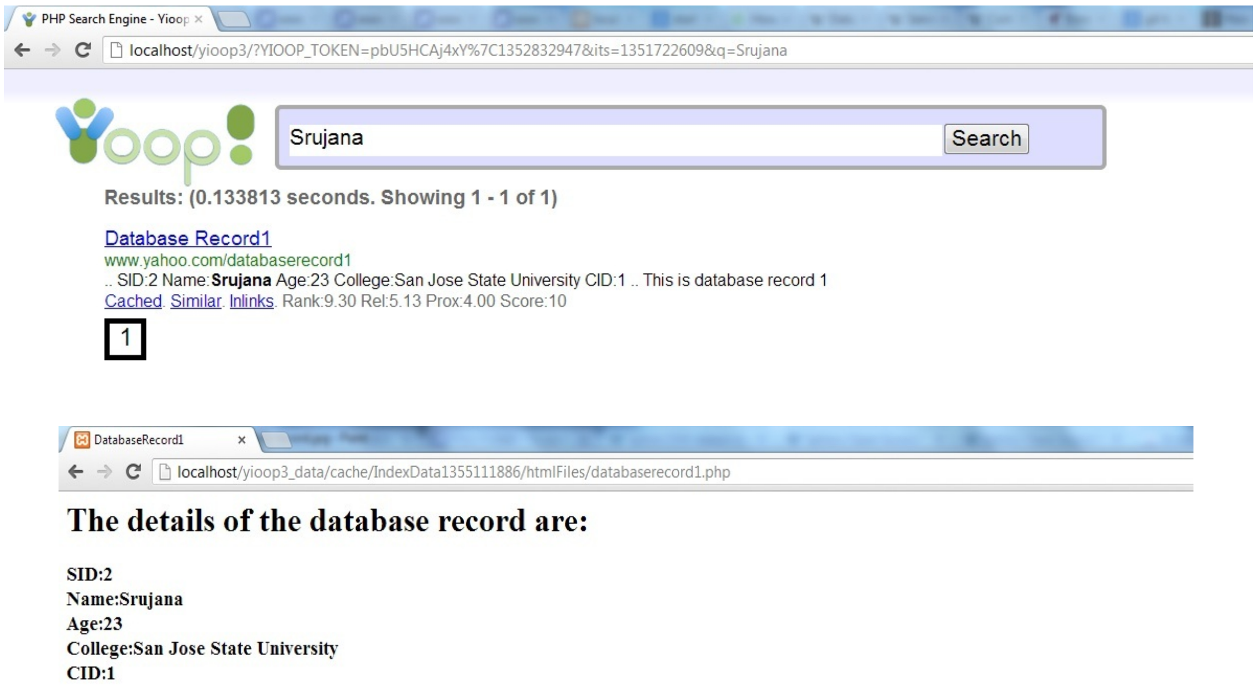
Edit Crawl Options

Crawl or Arc Folder to Re-index: ARCFILE::Database files

Database Connection Details

Hostname	localhost
Username	root
Password	
Database Name	database1
Query	select * from table1

Figure 23: Database connection details



localhost/yioop3/?YIOOP_TOKEN=pbU5HCAj4xY%7C1352832947&its=1351722609&q=Srujana

Srujana

Results: (0.133813 seconds. Showing 1 - 1 of 1)

Database Record1
www.yahoo.com/databaserecord1
.. SID:2 Name:**Srujana** Age:23 College:San Jose State University CID:1 .. This is database record 1
Cached. Similar. Inlinks. Rank:9.30 Rel:5.13 Prox:4.00 Score:10

localhost/yioop3_data/cache/IndexData1355111886/htmlFiles/databaserecord1.php

The details of the database record are:

SID:2
Name:Srujana
Age:23
College:San Jose State University
CID:1

Figure 24: Search results for databases

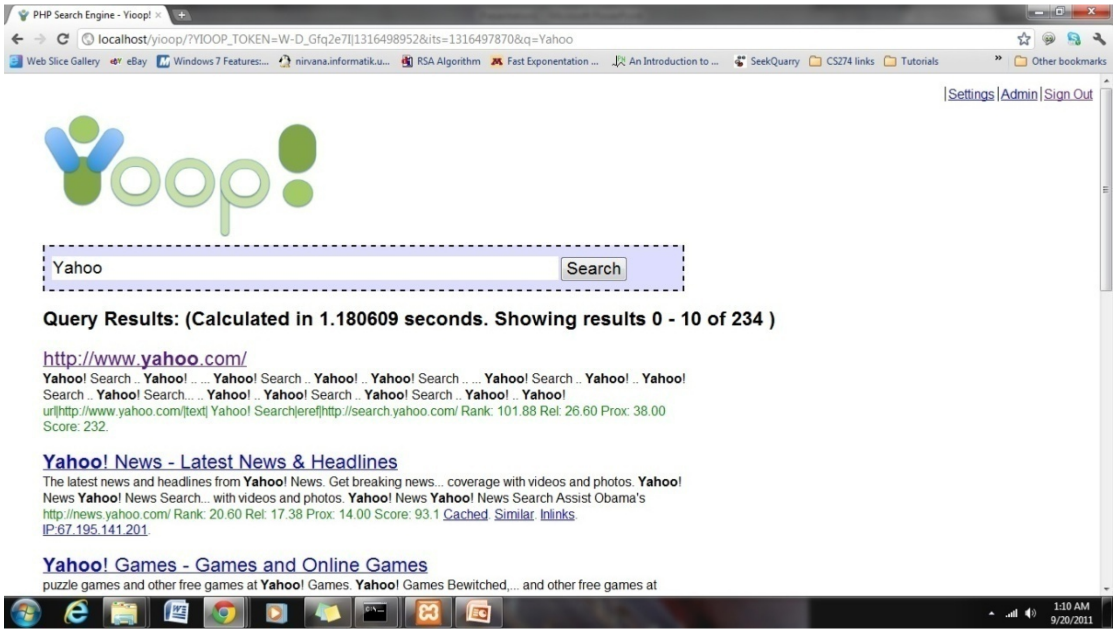
5 Testing and Results

This section describes the testing performed in order to test the usability of the user interface developed in Yioop, to test the efficiency of the results and also displays the test results. It is divided into three sub sections; URL shortening testing, usability testing of the interface, efficiency of the results. In order to test if Yioop is treating the short links correctly, crawling has been done on the short links and the results obtained were observed. The testing of crawling and indexing of log files and database files in Yioop included both usability testing and testing the efficiency of the implemented features. The testing methods and results are also described in this section.

5.1 URL Shortening

Testing for this feature is done in order to test if Yioop works correctly with short links. It checks to see if the results improved after the code is embedded into Yioop.

Firstly, a crawl is performed on a bitly link. This crawl is performed before applying the patch and also after applying the patch. The results were compared and it showed that Yioop improved on the results displayed. Also, the short links were avoided and redirected to the original links and these links were crawled and indexed.


PHP Search Engine - Yioop!
localhost/yioop/?YIOOP_TOKEN=W-D_Gfq2e7I|1316498952&its=1316497870&q=Yahoo
|Settings|Admin|Sign Out
Yahoo
Search

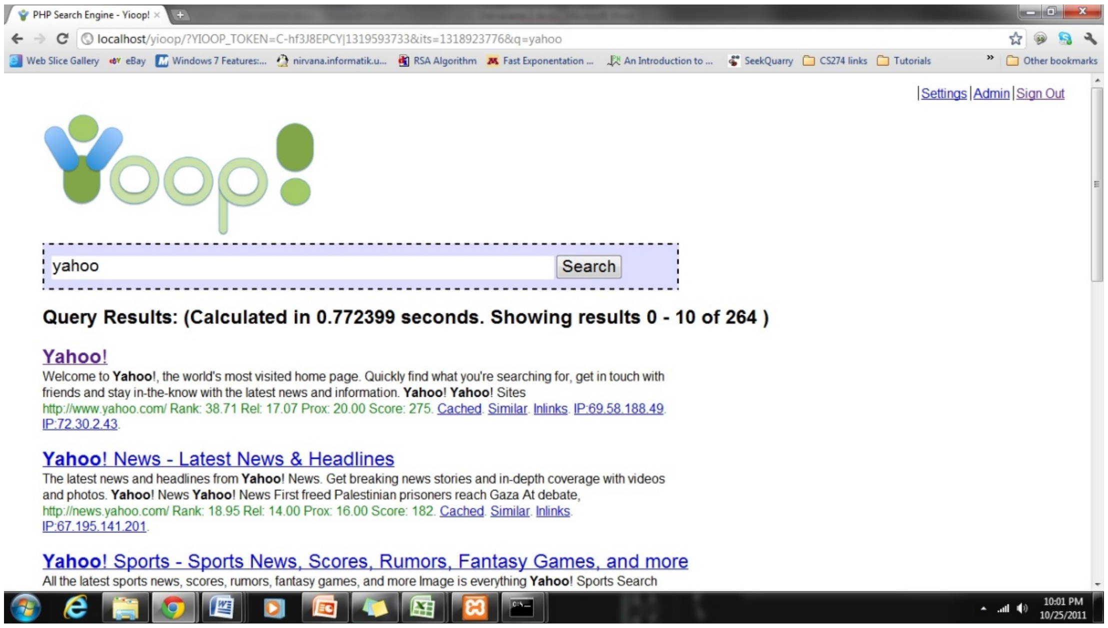
Query Results: (Calculated in 1.180609 seconds. Showing results 0 - 10 of 234)

http://www.yahoo.com/
Yahoo! Search .. Yahoo! .. ... Yahoo! Search .. Yahoo! .. Yahoo! Search .. ... Yahoo! Search .. Yahoo! .. Yahoo! Search .. Yahoo! Search... .. Yahoo! .. Yahoo! Search .. Yahoo! Search .. Yahoo! .. Yahoo!
url|http://www.yahoo.com/|text| Yahoo! Search|eref|http://search.yahoo.com/ Rank: 101.88 Rel: 26.60 Prox: 38.00 Score: 232.

Yahoo! News - Latest News & Headlines
The latest news and headlines from Yahoo! News. Get breaking news... coverage with videos and photos. Yahoo! News Yahoo! News Search... with videos and photos. Yahoo! News Yahoo! News Search Assist Obama's
http://news.yahoo.com/ Rank: 20.60 Rel: 17.38 Prox: 14.00 Score: 93.1 Cached. Similar. Inlinks.
IP:67.195.141.201.

Yahoo! Games - Games and Online Games
puzzle games and other free games at Yahoo! Games. Yahoo! Games Bewitched,... and other free games at

Figure 25: Results before the addition of code


PHP Search Engine - Yioop!
localhost/yioop/?YIOOP_TOKEN=C-hf3J8EPCY|1319593733&its=1318923776&q=yahoo
|Settings|Admin|Sign Out
yahoo
Search

Query Results: (Calculated in 0.772399 seconds. Showing results 0 - 10 of 264)

Yahoo!
Welcome to Yahoo!, the world's most visited home page. Quickly find what you're searching for, get in touch with friends and stay in-the-know with the latest news and information. Yahoo! Yahoo! Sites
http://www.yahoo.com/ Rank: 38.71 Rel: 17.07 Prox: 20.00 Score: 275. Cached. Similar. Inlinks. IP:69.58.188.49. IP:72.30.2.43.

Yahoo! News - Latest News & Headlines
The latest news and headlines from Yahoo! News. Get breaking news stories and in-depth coverage with videos and photos. Yahoo! News Yahoo! News First freed Palestinian prisoners reach Gaza At debate,
http://news.yahoo.com/ Rank: 18.95 Rel: 14.00 Prox: 16.00 Score: 182. Cached. Similar. Inlinks.
IP:67.195.141.201.

Yahoo! Sports - Sports News, Scores, Rumors, Fantasy Games, and more
All the latest sports news, scores, rumors, fantasy games, and more Image is everything Yahoo! Sports Search

Figure 26: Results after the addition of code

As seen in the above figures, before the addition of code, Yioop displayed the Yahoo URL as a plain text in the title of the page. After the addition of code, Yioop had redirected the bitly link correctly and the Yahoo link is displayed as the title and as the first result.

Secondly, another crawl was done in Yioop with the Yioop's default sites and domains. A search was done on the word "IBM". The results obtained before the patch and after the patch differed, the results improved and were much clearer. The results of this crawl can be seen in Figure 18.

5.2 Usability testing

Usability testing is a technique used in the field of user interface design or user centered design to evaluate a product by testing it on users [5]. This practice is very much used while building various applications in order to measure the amount of satisfaction provided to the users. It is also done to check if the product is able to perform well in those areas for which it is designed [5]. This is one of the best practices as it involves the users directly to evaluate the design [5].

Usability testing is done to measure to what extent the user is satisfied in the following four subject areas :

- **Efficiency -** The time taken or the number of steps taken for a user to complete a particular task [5].
- **Accuracy -** The number of mistakes made by a user while performing a particular task [5]. It can be calculated as a measure of the number of mistakes made a particular user by a predefined assumed number of mistakes that can be made by the user.
- **Recall -** This measure refers to the amount of work that a user could recall after a period of time concerned with a particular task [5]. It can be measured in the amount of time that the user takes to complete the task after a period of time. If the amount of time decreases for a user, it can be said that the recall measure is improved.

- **Emotional Response -** This refers to the amount of satisfaction and the kind of emotion felt by a person while performing a particular task [5]. It can be measured by providing a scale of rating and asking the user to rate the task in terms of the satisfaction gained.

The feature of crawling and indexing of log files and databases in Yioop included the creation of new user interface for the user to enter details on the web interface of Yioop. In order to test the usability of this interface testing was performed with five users whose age lies between 22-26 years. The users chosen were familiar with the usage of computers and also possessed some prior knowledge on the basic search engine and crawling concepts. In order to perform testing, there were seven tasks that the user was asked to perform. The seven tasks were assigned in such a way that they possess the same difficulty level and can be completed in the same amount of time. For these purposes the resources given to the users were the location to download Yioop, documentation of Yioop, log files, a database and a brief explanation of this project.

The tasks given to the users involved both working with Yioop and also tasks related to this project. This is done in order to gain a comparative analysis between both of them and also to see if the features embedded in Yioop are compatible enough to work with Yioop.

Task1 : Download and Install Yioop (Includes downloading and installing Xampp)

Task2 : Configure Yioop

Task3 : Start a new crawl in Yioop

Task4 : Set up log files folder in Yioop for archive crawling

Task5 : Set up a folder for database in Yioop

Task6 : Input details into log files interface, save them and start a crawl with them

Task7 : Input details into database interface, save them and start a crawl with them

The user was being observed while performing all the above functions and questions were asked in order to measure the effectiveness of the interface in all the four subject areas of usability testing mentioned above.

(1) Efficiency

The efficiency is calculated in terms of the time taken to complete a particular task for a particular user [5]. The time taken for each user to complete each task is recorded and the results are shown in the graph in Figure 27. Most of the users took the same amount of time to complete each of the tasks. Few of them found issues in order to start a crawl when they were starting a crawl using the manage machines section of Yioop and some of them spent time reading the documentation of how to achieve certain tasks.

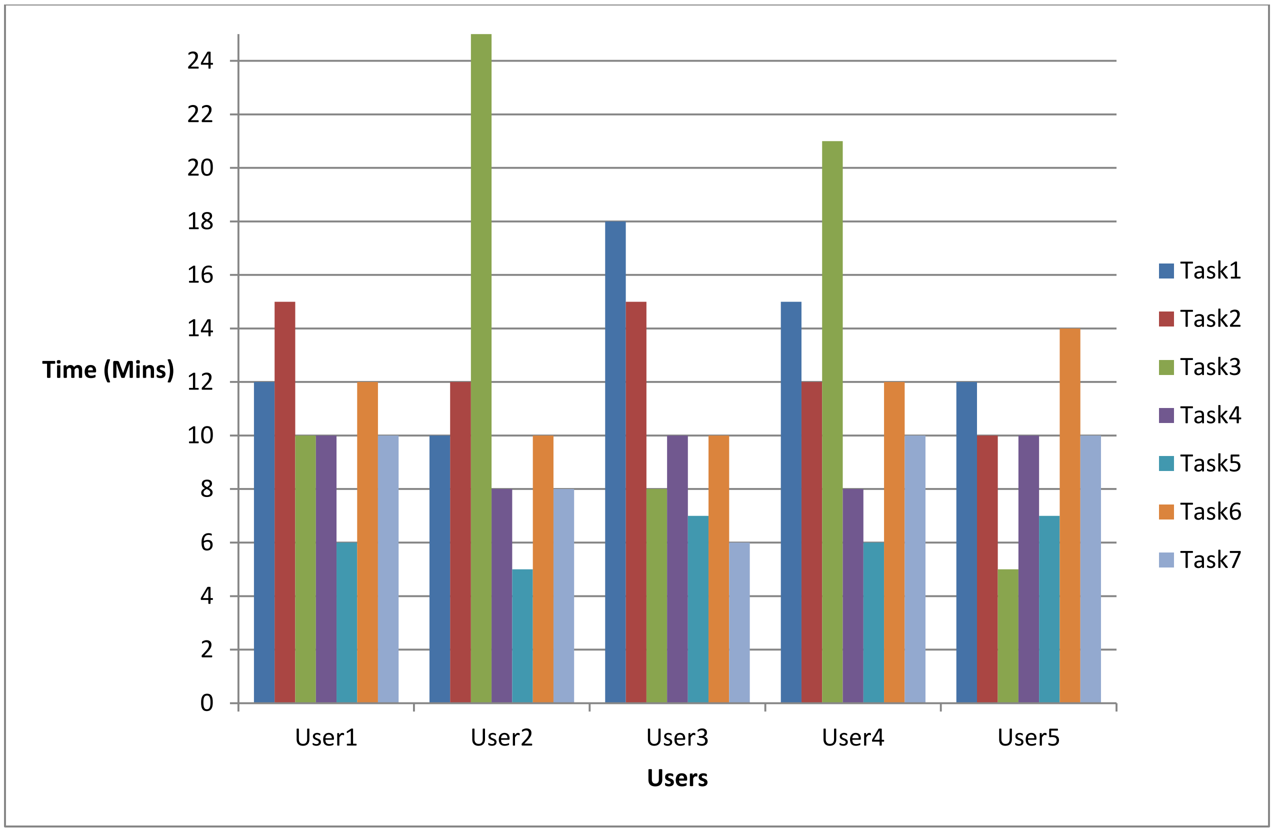


Figure 27: Time taken for each user to complete a task

The average amount of time taken for each of the tasks by the users is shown in the graph shown below. It gives a good comparison between the amount of time taken by normal Yioop processes and the new features implemented. The graph shows that the user takes comparatively the same amount of time taken for the normal Yioop processes which shows the new interface developed is efficient and compatible enough with Yioop.

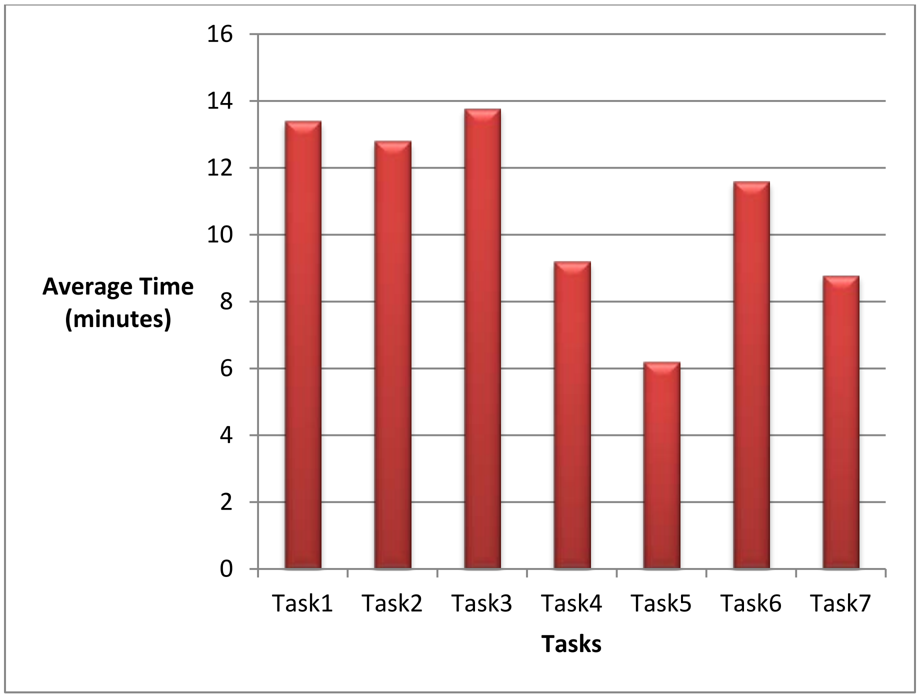


Figure 28: Graph displaying the average time taken by users for each of the tasks

(2) Accuracy

In order to test the accuracy of the users performing the tasks, the average number of mistakes made by the users for a particular task are calculated. A certain number of mistakes are assumed to take place for every task. The average number of mistakes done are compared over the assumed number of mistakes and the accuracy percentage is calculated. The table below shows the assumed number of mistakes for each of the tasks assigned to the user. A graph is also plotted in order to show the percentage of accuracy for each task. This is calculated as the percentage of average number of mistakes by users for a particular task by the assumed number of mistakes of each ask.

Tasks	Assumed number of mistakes
Task 1	2
Task 2	3
Task3	5
Task 4	3
Task 5	4
Task 6	4
Task 7	2

Table 1 : Assumed number of mistakes for each task

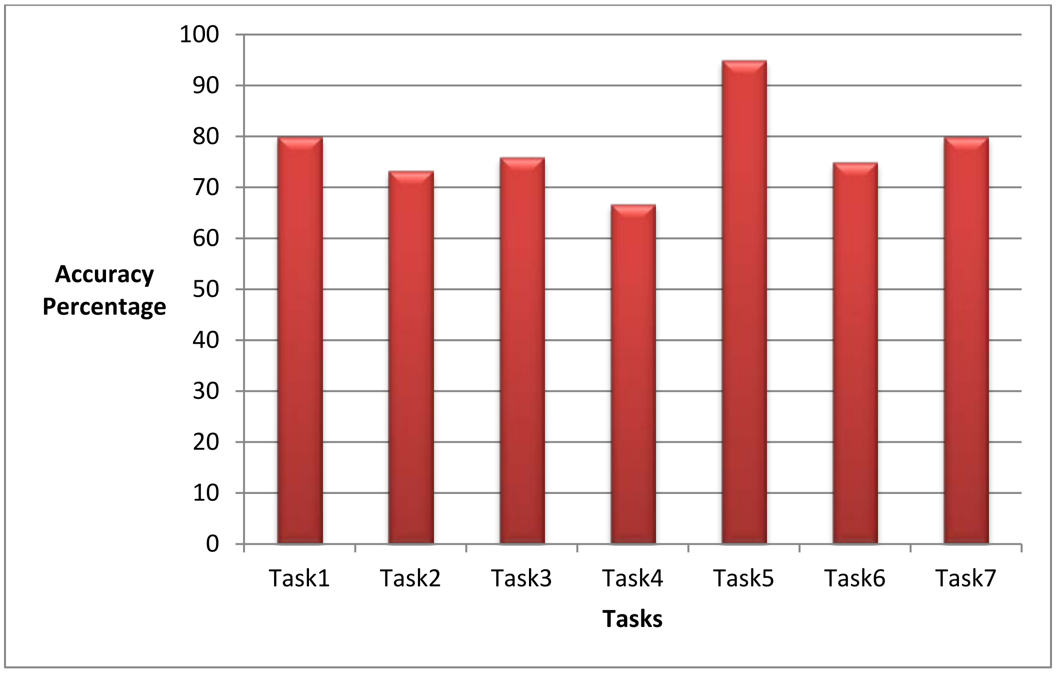


Figure 29: Accuracy Percentage of tasks performed by user

(3) Recall

Recall is the area where testing is done on order to measure the amount of work that could be recollected without the need of referring back to the previous work done [5]. This measure can be calculated by measuring the amount of time taken by the user to do the tasks for the first time and the second time and comparing them [5]. A decrease in the amount of time shows improvement in the tasks performed.

In this project, a gap of five days has been given to the users and testing was performed again asking the users to perform the same tasks as before. The times taken for the first time and the second time are plotted and there is an improvement in the times. The users performed few tasks much faster than before. There were very few instances of some of them taking a longer time due to some issues with their systems. Most of them claimed that they could remember a large amount of work done before and only few minute things needed to be referred again. The graph below shows the average time taken by users for the same tasks for the first time and the second time.

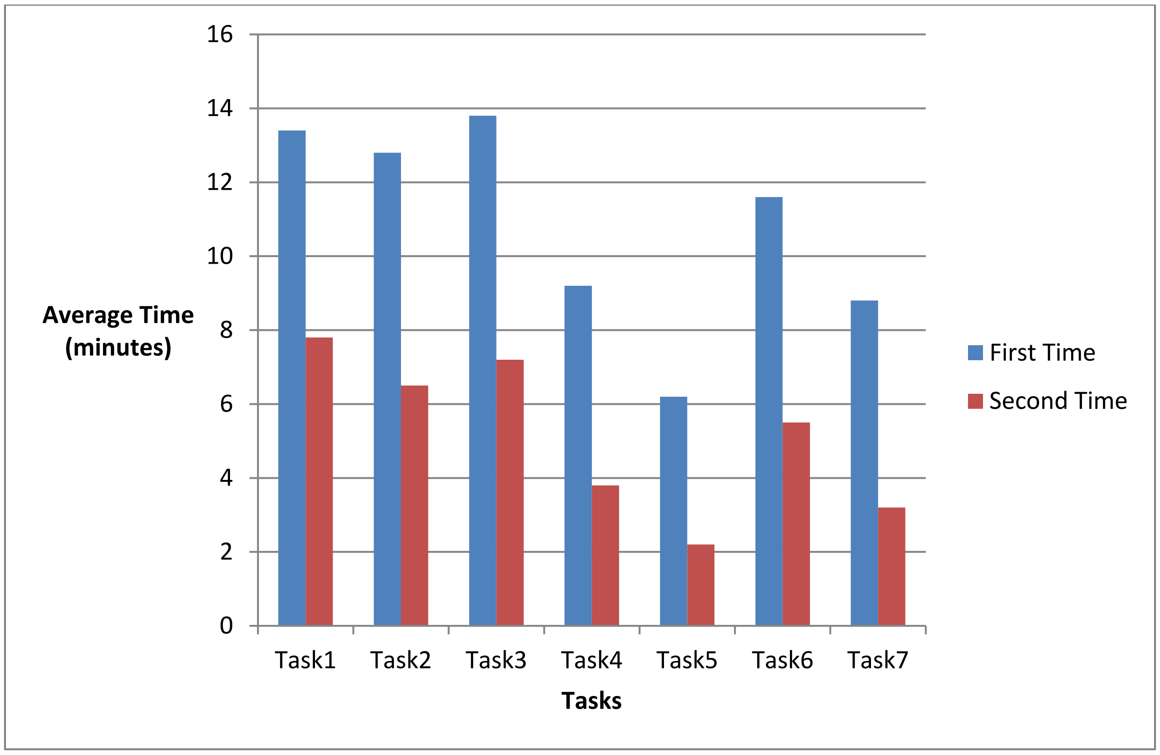


Figure 30: Graph showing the amount of time taken by users for the first time and second time

(4) Emotional Response

In order to measure the emotional response of the users, questions were being asked during the time when they were accomplishing the tasks. Each user was asked to give a rating based on their satisfaction levels. The rating was on a scale of 1 to 10, where 1 is completely dissatisfied and 10 is completely satisfied. The satisfaction factor includes the ease with which the user could use the interface, how confident the user feels about repeating the steps after a period of time and also the feel of the interface. A graph is plotted to show the average rating of satisfaction of users on a scale of 10 for each of the tasks performed. The results show that the users are satisfied with Yioop and the new interface to a lot of extent.

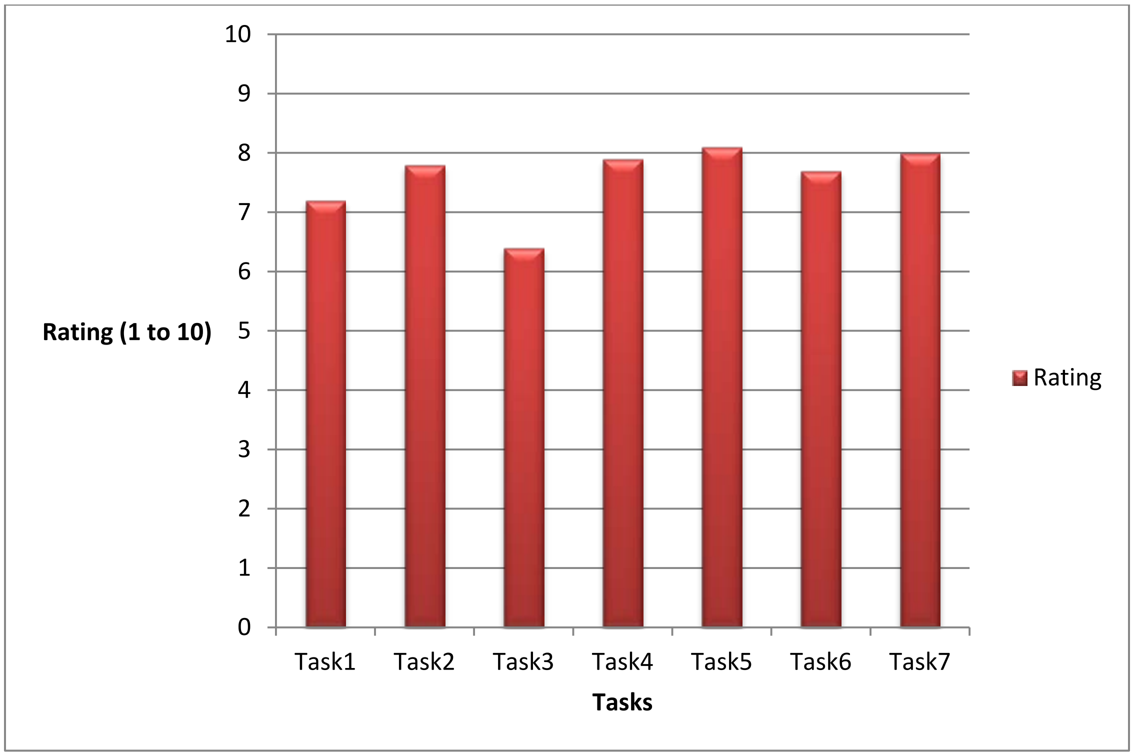


Figure 31: Average rating of satisfaction of user

5.3 Testing efficiency

Testing efficiency of the written code is one of the most important consideration in a project. In order to test the efficiency, few types of tests were performed. The first test is performed by measuring the amount of time taken to crawl and index the log records and the second test is performed to check how the code is working when different kinds of queries are given in the database interface. Also, performance testing was done by testing the query time that is taken by Yioop in order to return the results for a search query versus the query time taken in MySQL to return the records using the LIKE operator. The testing using the LIKE operator was performed as it is the most commonly used method to perform a text based search if a database indexer is not available. Therefore, the query times taken by the LIKE operator and Yioop would give a good comparison between both of them and provide a good picture on the performance of Yioop with the new features implemented. Also, the number of records returned are also recorded.

Firstly, the code written for log records was tested by varying the number of records ranging from 100 record to 100,000 records. These crawls were performed using a single machine with a single server. The times taken for each of these crawls is recorded. It was noted that the time taken to crawl and index the log records increased gradually with the increase in the number of records. A graph was plotted to show the exponential growth of time as the number of log records increases.

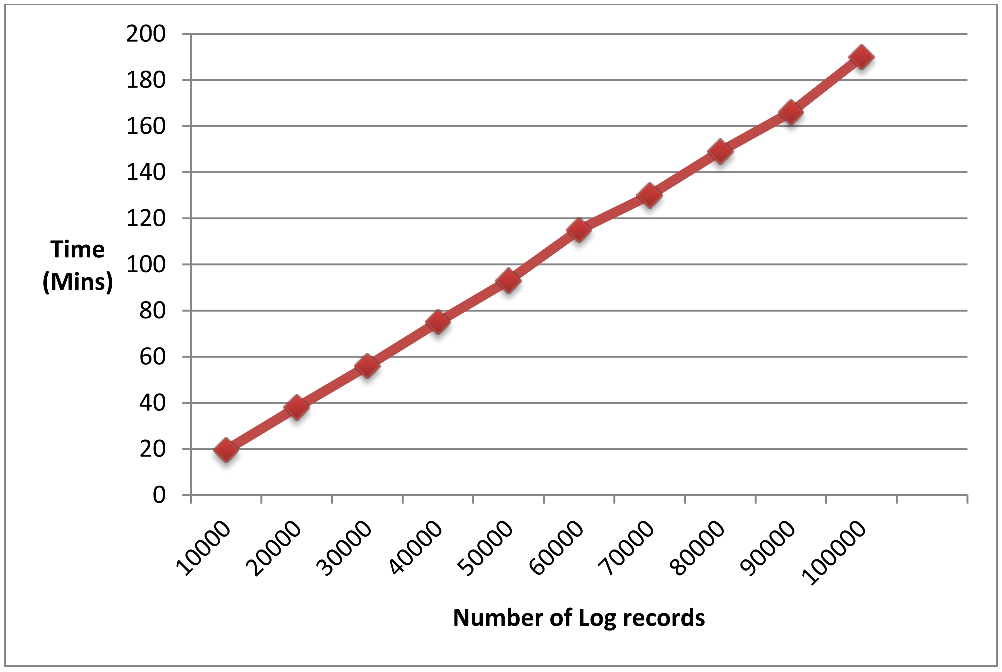
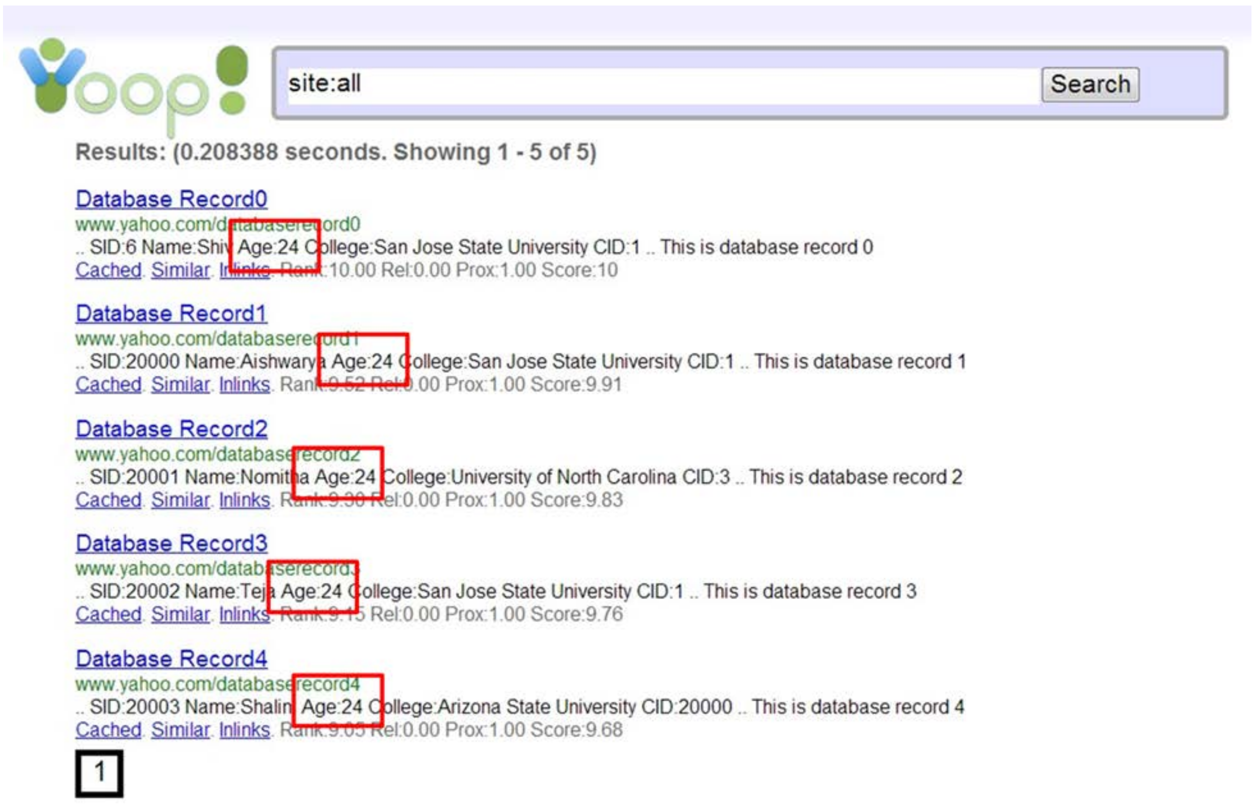


Figure 32: Straight line graph of time versus the number of log records

Secondly, testing was done for the databases code. In order to test the correctness of code, testing was done by providing different queries in the database interface and checking if the records are limited and extracted based on the queries inputted. The database table used for this purpose is populated with 20000 records by using a PHP script. The first query inputted was to query the entire table whereas the second query inputted was to chose and crawl only those records in the table whose age field is 24. For the second crawl, the records with age 24 are only crawled and all the other records in the table are ignored. The search results of the first crawl is the same as the one shown for database records in Figure 24. The results of the second crawl are shown below.


site:all

Search

Results: (0.208388 seconds. Showing 1 - 5 of 5)

Database Record0
www.yahoo.com/databaserecord0
.. SID:6 Name:Shiv Age:24 College:San Jose State University CID:1 .. This is database record 0
Cached. Similar. Inlinks. Rank:10.00 Rel:0.00 Prox:1.00 Score:10

Database Record1
www.yahoo.com/databaserecord1
.. SID:20000 Name:Aishwarya Age:24 College:San Jose State University CID:1 .. This is database record 1
Cached. Similar. Inlinks. Rank:9.52 Rel:0.00 Prox:1.00 Score:9.91

Database Record2
www.yahoo.com/databaserecord2
.. SID:20001 Name:Nomitha Age:24 College:University of North Carolina CID:3 .. This is database record 2
Cached. Similar. Inlinks. Rank:9.30 Rel:0.00 Prox:1.00 Score:9.83

Database Record3
www.yahoo.com/databaserecord3
.. SID:20002 Name:Teja Age:24 College:San Jose State University CID:1 .. This is database record 3
Cached. Similar. Inlinks. Rank:9.15 Rel:0.00 Prox:1.00 Score:9.76

Database Record4
www.yahoo.com/databaserecord4
.. SID:20003 Name:Shalini Age:24 College:Arizona State University CID:20000 .. This is database record 4
Cached. Similar. Inlinks. Rank:9.05 Rel:0.00 Prox:1.00 Score:9.68

1

Figure 33: Crawl results for second query

Thirdly, testing was performed to compare the performance of Yioop with MySQL LIKE operator. This testing was performed with the LIKE operator as it is the default method used to index if a database indexer is not available. The method of testing is described in this section.

In order to perform this test, a table with 2000 records is created in a database. The table contained four fields namely, company ID, company name, number of employees and a description about the company. The description field of this table is of main interest in this particular test. An index has been created in Yioop to query the table, i.e., the database indexing in Yioop feature was used to index the entire table. The query given to Yioop in the user interface of Yioop was "SELECT * FROM TABLE4", where table4 is the table having all the company details. This crawl was set as index in Yioop to search for various search queries.

Testing was performed by giving various search queries in the Yioop search interface and searching for the same search queries in MySQL using the LIKE operator. The various search queries were "on demand", "popular", "Fortune 500", "best companies", etc. These search queries were entered in the Yioop search interface and also in MySQL with a query using the LIKE operator. The query used in MySQL was "SELECT * FROM TABLE4 WHERE DESCRIPTION LIKE '%search query%'". The search query was replaced with the search queries that are mentioned above. All the query times taken in Yioop and MySQL are recorded and also the number of records returned by each of them are recorded. A graph was drawn in order to give a comparison between both of them. The graph is shown below.

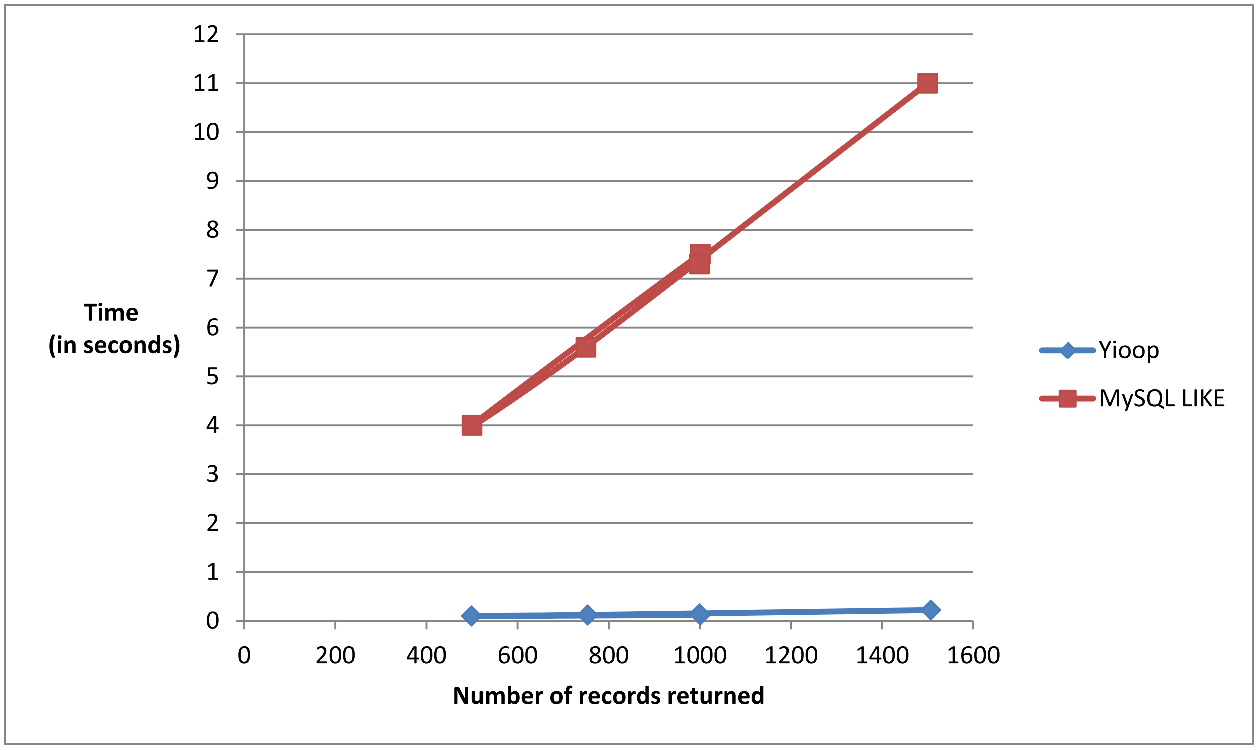


Figure 34 : Comparison of Yioop with Mysql LIKE operator

The results obtained in this test show that the performance of Yioop is better than the performance of the MySQL LIKE operator. The time taken for LIKE to return all the results for the search queries was higher than the amount of time taken for Yioop.

Therefore, the test results in this section showed the code works efficiently and the user interface is satisfactory for the users.

6 Conclusion

Often, users might want to crawl and index different file formats and be able to search for keywords in them. These kind of file formats that are often not crawled and indexed by traditional search engines fall into the category of invisible web. In this project, an implementation has been done to add to Yioop the abilities to crawl and index log files and database content.

In order to make it possible for Yioop to accomplish crawling and indexing of log files and databases, the existing concept of archive crawling in Yioop was used. There were two new archive bundle iterators added to Yioop, namely, LogArchiveBundleIterator and the DatabaseArchiveBundleIterator. Firstly, a user interface is designed for this purpose to help the user in customized crawling. For log files, the users can specify their own field names and also select a field type from the predefined field types provided to them for the log fields. Secondly, for database indexing, the user can limit the number of records depending on the query entered by the user. This feature in database indexing provides the user with a greater personalized crawling experience. Implementation was done to retrieve the records and create web pages for each of these records so that Yioop could crawl and index them. A search was done on few keywords which were known to be present in the records and the search results were displayed appropriately in Yioop search interface.

There was extensive testing performed in terms of efficiency of the user interface built for log files and databases and also for the backend code. The usability testing has been done in four areas, namely, efficiency, accuracy, recall and emotional response. The time taken by the users, their ratings and feedback given have been recorded and plotted in terms of graphs. In order to test the efficiency of the backend code in terms of time taken to index and crawl records, 100,000 records were generated and Yioop was made to crawl and index all the records. Also, different kinds of queries were given to the database interface to test if the records were being retrieved appropriately. The results of the testing phase were recorded and were displayed graphically in the testing section of the report.

Also, as a modification to the Yioop search engine, code was added in order to make Yioop handle short URLs in a more meaningful way as explained in the previous sections of this report. Yioop provided the necessary framework for the implementation of this project as a reason of its archive crawling feature. In order to improve this project, a study can be done on how the page relevance is calculated for the log records and database records in Yioop. Presently the page relevance is being calculated using the algorithm in Yioop. Based on the study, the search results can be improved by implementing a new algorithm to calculate the page relevance of these records. This could improve the search results of the records. This can be considered as a potential future work for this project.

7 References

1. Pollett, C. (2012). Yioop! Documentation v 0.90. Retrieved from http://www.seekquarry.com/?c=main&p=documentation

2. Peng, Wei., Tao, L., & Ma, S. (2005). Mining logs files for data-driven system management. *ACMSIGKDD Explorations Newsletter - Natural language processing and text mining, 7*(1).

3. He, B., Patel, M., Zhang, Z., & Chang, K.C. (2007). Accessing the Deep Web. *ACM, 50*(5), 94-101.

4. Deep Web. (2012, November 9). In *Wikipedia*. Retrieved from http://en.wikipedia.org/wiki/Deep_Web

5. Usability Testing. (2012, October 23). In *Wikipedia*. Retrieved from http://en.wikipedia.org/wiki/Deep_Web

6. Bailey, B. (2006, March). Getting the complete picture with Usability Testing. Retrieved from http://www.usability.gov/articles/newsletter/pubs/030106news.html

7. Lewandowski, D., & Mayr, P. (2006). Exploring the academic invisible web. *Library Hi Tech, 24*(4), 529-539.

8. Sphinx Documentation. (n.d.). In *Sphinx* website. Retrieved from http://sphinxsearch.com/docs/

9. Server Log. (2012, July 18). In *Wikipedia*. Retrieved from http://en.wikipedia.org/wiki/Server_log

10. Destailleur, L. (2008, December). AWStats. Retrieved from http://awstats.sourceforge.net/

11. Log Files. (2012). In *Apache* website. Retrieved from http://httpd.apache.org/docs/1.3/logs.html

12. Databases. (2012, November 26). In *Wikipedia.* Retrieved from http://en.wikipedia.org/wiki/Database

13. Zillman, P. M. (2012, November 1). Deep Web Research and Discovery Resources 2012. *Virtual Private Library*.

14. URL Shortening. (2012, November 23). In *Wikipedia.* Retrieved from http://en.wikipedia.org/wiki/URL_shortening

15. Sebastian. (2008, October 20). Crawling Vs Indexing. Retrieved from http://sebastians-pamphlets.com/crawling-vs-indexing/

16. Web Search Engine. (2012, November 25). In *Wikipedia.* Retrieved from http://en.wikipedia.org/wiki/Web_search_engine

17. Open Source Search Engine Software. (2012). In *SeekQuarry*. Retrieved from http://www.seekquarry.com