A Study of UnIndexed Web

Doga Tuncay (dtuncay@umail.iu.edu)
Chi-yang 'Eric' Tsao (tsaoc@indiana.edu)
Arvind Dwarakanath (adwaraka@indiana.edu)

Department of Computer Science
Indiana University, Bloomington

Project Abstract

The project deals with the study of the unindexed web. In brief, the unindexed or 'UnWeb' as we call it in the project are those pages on the internet that are not indexed (usually that will be by the host site) and are therefore not crawled by a search engine. The project tries to establish the relationship between the unindexed portions of the web and the indexed portions and determines how a user navigates from the unindexed of the web to the indexed portion and vice-versa.

Initially, we talk about the motivation and the challenges which we faced in the project. Furthermore, we discuss the schema used in storage of the data and the tables that were deemed necessary and the codes that were implemented for the collection and the querying of the data. Also, we discuss a few queries that may be fired on the schema to retrieve the important information visually or as a simple result set.

Motivation of the Project

Generally, search engines retrieve a set of indexed web pages and make them available to the users. However, the web contains a large number of unindexed web pages; something that a search engine cannot index. These web pages exist due to human actions such as opening a new mail account, filling a form or a survey, a private blog etc. So this research overall will enable us to understand how we can access the indexed portions of the web from unindexed ones. This project specifically focuses on the storage and analysis of the graphed data of URLs.

Goal of the Project

The goal of the project is to design and implement optimal data storage model for the data. The data is stored as a URL descriptor and as a digraph to represent the links uniquely. Also, we needed to determine how to efficiently access and retrieve data.

Challenges Faced

The amount of data that we store is very large and is linked together in a very complicated manner. So the main characteristics which we aim for in our backend database are the scalability, performance and the cost of maintenance. The project also demands at times that we may need to determine the relationship between two entities. In such cases, the implementation has increased in complexity and in data size as well. So this was a major challenge in implementing the project and others alternatives may have to be explored or we can make optimizations to the existing storage.

Related Research

The unindexed web is a topic of interest for a lot of organizations, specifically with those that deal with the design and the implementation of the search engines. However, their goal is to increase the effectiveness of their search engines rather than analyze the effect of their presence. The number of works that actually analyze their behaviour is very limited.

The closest work that has come near to our project is a paper title '*Web Mining and its SQL based Parallel Execution*'. This paper talks about web-mining and parallel querying, but the essence of the paper is splitting the data initially into Web Usage and Web Structure and then mining the required category for information. Again the paper disregards the concept of unindexed web pages, but not completely.

In addition to storage, the papers mentioned in the references section namely the BFS search and the Graph structure of the web was of great importance in the querying.

Tools Required and Storage Analysis

The proposal of the project mentioned certain tools that were needed to collect the data for storage and analysis. We enumerate them for the reader's benefit.

- URL Crawler tool – utilizes a tool that scans all the reachable URLs given a root node upto a predetermined level.
- Human driven Events analysis – using this tool in combination with an http analyzer returns the URLs that were traversed by a user in a session.
- In addition we have a tool called Scraper that identifies whether a particular web-page accessed is an indexed page or not. This tool refers Microsoft Bing Search Engine for verifying the same.

These above tools enabled the collection of data. The human-driven events are mapped over a graph of web pages and the URL crawler establishes a sub-graph germinating from the accessed URL. This enables the study of what a user accesses and what kinds of web-pages surround the accessed URLs of a user. In addition, we also establish what the web-pages are using the scraper: indexed or unindexed?

Storage of Data – Our initial thoughts were to utilize the RDF framework for storage of the data. But the number of values to be stored for even the URL crawler even for one root URL for a very low level was quite large (numbers ran upto approx 1 millions). So for the effective storage, a semantic database was considered and declared not feasible as the amount of data would break any open-source RDFS database.

The initial experiments were being carried out in MySQL Relational database and the linking information was stored in the MySQL database as a separate table.

Considering the fact that we stored the human events and the graphed data, we needed to break down the problems into two sections; storing the crawler data and then the human driven events. Each section is discussed below in detail.

Storage Design of Crawler Data

The storage of Crawler Data takes place in two phases – storage of the URL details and then the storage of the link i.e. the parent-child relationship. The URL details show the details of the URL accessed. The storage of the link details how URL <http:/x> is a parent of <http:/y> and the access code that is returned when the access is made.

The URL details are stored in the ‘url_crawler’ table (*Fig 1*) and the link details are stored in the ‘Link’ table (*Fig 2*). The schema details as executed in the JDBC/MySQL are as follows:-

```
                    The ‘url_crawler’ table schema

CREATE TABLE url_crawler
(   id unsigned INT NOT NULL AUTO-INCREMENT PRIMARY KEY,
    URL VARCHAR(700),
    Secured VARCHAR(10),
    tld VARCHAR(4),
    index INT(1)    );
```

Note: The URL data type is unique so that we don’t store the same URL’s details multiple times.

Fig 1: The url_crawler table schema

```
                    The ‘Link’ table schema

CREATE TABLE Link
(     id_1 INT,
      id_2 INT,
      http_access INT,
      FOREIGN KEY id_1 REFERENCES url_crawler(id),
      FOREIGN KEY id_2 REFERENCES url_crawler(id),
      PRIMARY KEY(id_1, id_2)
);
```

Note: The Link value (child, parent) is unique.

Fig 2: The Link table schema

The url_crawler table consists of the URL name that is extracted from the URL crawler data. Based on the URL name, the values of the top level domain, the type of protocol used will be extracted. The value for the 'index' field will be determined using the Scraper code. Initially both the crawler and the HDE codes hardcode the value as unindexed (that is value 0). Then we execute the scraper to determine the index value. Thereafter, we change the value to indexed i.e. 1 once the Scraper finds a URL link to be indexed. Each URL stored in the database is unique and so they will each be indexed by an identity field integer which will be auto-incremented.

The 'Link' table, as mentioned above, stores the individual link. Rather than storing the URL Link, it was thought to be more space efficient to extract the corresponding id of each URL id to identify each URL rather than storing the entire URL link. Therefore care must be made to ensure that the value of (id_1, id_2) remains unique. Please note primary (id_1, id_2).

Storage Design of Human-Driven Events

The human-driven events utilizes a HDE generating tool that extracts the required information of how a user accesses web-pages. This is basically a sequence of web-pages that are accessed and the information that is extracted along with the web-page accessed. The information includes the process, the type of web-page, the host and other information. Generally, all of it is not required. The Human-driven events table contains the following fields as in (*Fig 3)*.

The table will contain the event_id that will be the *unique identifier* to the URL. Each session will generate an individual file; so each session will be identified using a session_id. The sequence Id will increment for each URL in the sequence within a session. The field's url_name are the URLS accessed and in case any are directed ones. Host and IP_address are optional.

Fig 4 shows the basic architecture of the project framework.

The 'Human-Driven Events' Table

Event_Id	Sequence_Id	Session_Id	url_name	Indexed	process	Host	IP_address
INT	INT	INT	VARCHAR	INT	VARCHAR	VARCHAR	VARCHAR

Fig 3: The above table that contains the human-driven events.

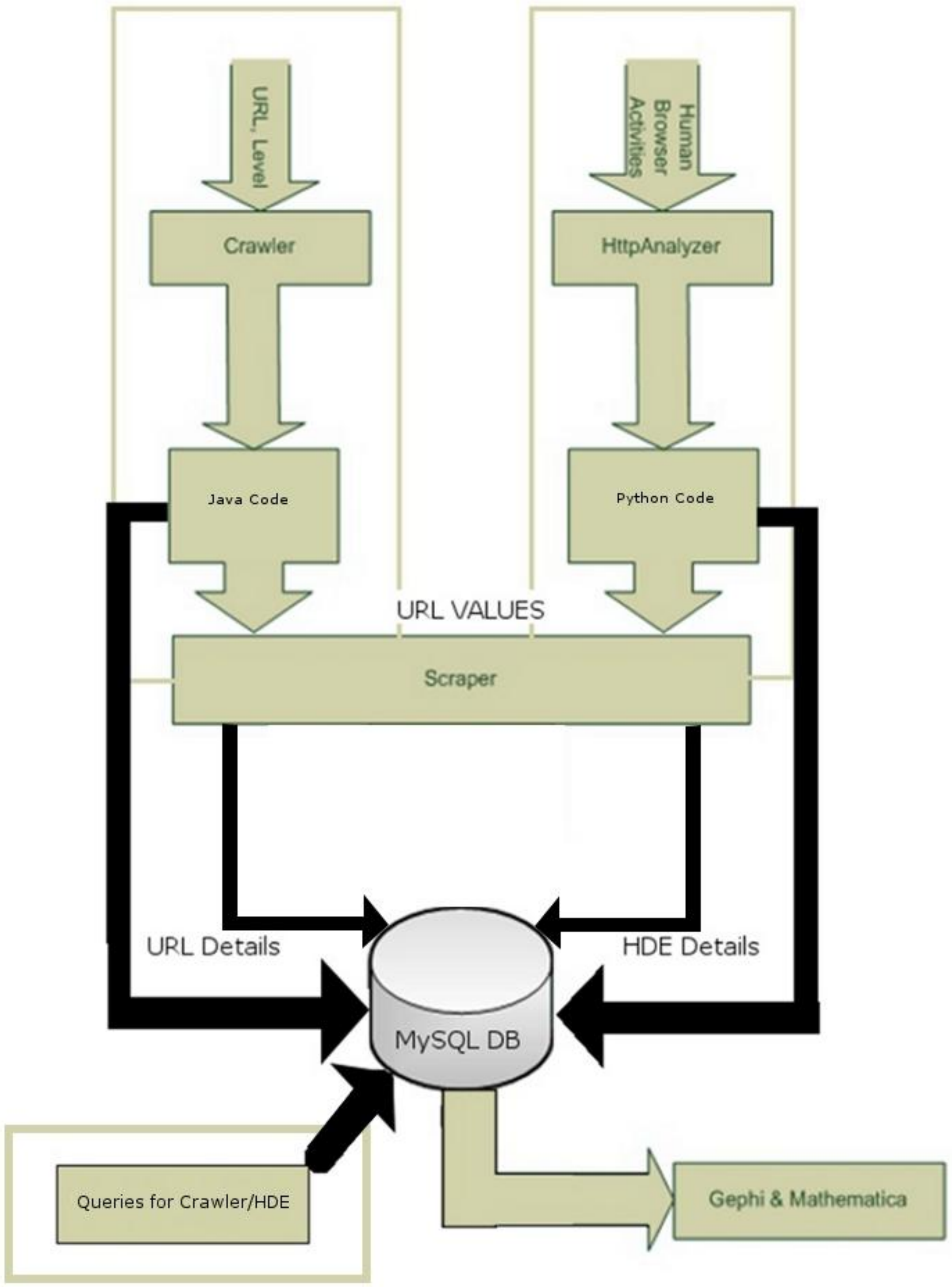

Fig 4: The architecture shows the basic framework for UnWeb. Note that the individual ochre boxes indicate the Crawler and HDE to be separate entities. So is querying for that matter.

Types of Queries & Examples

The above storages answer some of our basic queries- how the unindexed web works and how they are linked to indexed data and vice-versa? The path that is taken by the user unindexed or indexed is surrounded by a vast network of web-pages and that will determine the exposure of a user to indexed and unindexed web. There is a lot of information that are stored in unindexed web; some of them may be malicious; there may be possible security leaks in what a user may suppose to be secret in his/her unindexed web-page; overexposure to certain types of domains etc.

Some of the initial queries that were tested will include the following. Traversal graphs in combination with iterative SQL querying have been utilized to solve some of the following queries instead of SQL queries.

Based on analysis of the schema structure, the querying was divided initially into a generalized structure. The querying types included:-

- Connectivity Related
- Nearest-Neighbour
- Human Events Related
- Keyword Search (*included in HDE*)
- Links related

Certain demo examples are given for some of the above queries along with the visuals.

Connectivity Related

1) Given a URL ‘x’ as a source URL and another URL ‘y’ as a destination URL; execute a query to check whether for all outgoing links from ‘x’, ‘y’ is reached within a predetermined level? The level will be specified by users for our tests. The level will not exceed 4. The value should not exceed the url_crawler level. The inputs and the outputs will be a follows:-

Reason: - To find a linkage between two URLs. This can help in determining relationship between two nodes.

Input: - The source URL, destination URL and the level will be input.

Output: - The output will return ‘found’ if destination URL is found within the level. The path will be returned and the result will be visualized using Gephi as shown in Fig 5.

2) Given your URL = 'x', determine whether there is a redirect in the tree back to the initial URL. For each URL verify that current_accessed_url = your_url (3 currently is the level; hardcoded this time).

Reason: - If the input URL is an unindexed URL and in the crawler if there is a path back to the input URL, there can be a compromise in the security of the URL contents.

Input: - The input will be a URL.

Output: - The number of times a URL points back to the original input URL will be returned

Links Related

3) Given your URL = 'your_url', count the number of broken links (4XX) that occur in the tree for a predetermined level. This code has a level of 3. Again it should not exceed the url_crawler level.

Reason: - To check any disconnections in crawled graphs. *This can be extended to check any redirected links as well (3XX), but the current crawler does not store any parent in some cases. This was initially thought for 3XX links only.*

Input: - The source URL, destination URL and the level will be input.

Output: - The number of times broken links are found will be returned within a predetermined level.

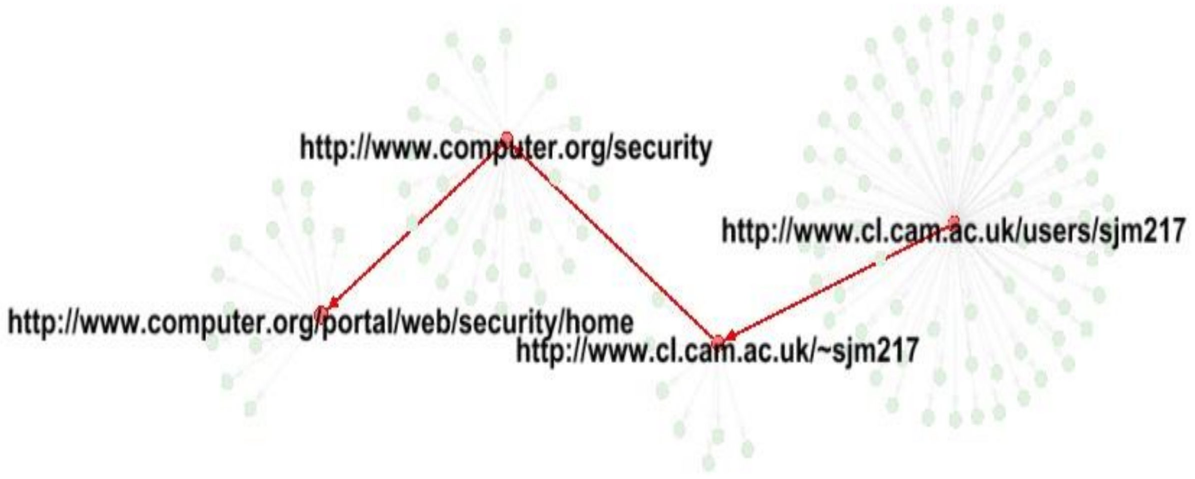
http://www.computer.org/security
http://www.cl.cam.ac.uk/users/sjm217
http://www.computer.org/portal/web/security/home
http://www.cl.cam.ac.uk/~sjm217

Fig 5: An example of Connectivity-Related query. The visualization is done using Gephi. The depth is level 3.

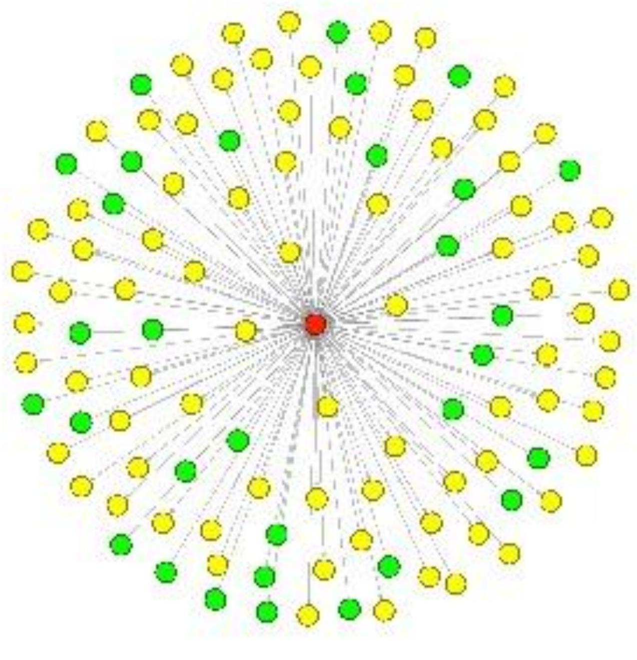

Fig 6: The Nearest Neighbour visualization. The colored nodes indicate inbound and outbound link URLs.

Nearest-Neighbour

4) Given a URL = <url_value>, determine all the nearest <URLs> till depth= url_crawler_ level that have indexed = 0 (unindexed).

Reason: - This will return a visual graph of the URLs which are currently the in links and the out links of a particular URL.

Input: - The source URL will be input.

Output: - The inbound links for the current data set and the outgoing links are displayed. Note that the inbound links will not be a complete set. It will be determined only by the current data set generated by the crawler and HDE.

Human-Driven Events

5) Finds the total number of indexed and unindexed URLs of all sessions and outputs the root information and show that root URL whether if it is searched using a search engine or not.

Reason: We want to find out how many indexed or unindexed webpages are being reached from search engines. This actually includes a portion of keyword search.

Input: All the sessions and URLs

Output: Number of indexed and unindexed of all sessions and the root information of those sessions.

6) Given a session in HDE data, find the consecutive URL pairs that are crawled , find the those pairs in the crawler table and retrieve the first level of childs of these 2 URLs, and find the common links and output the information of common links

Reason: We want to find out, how a user can access common links from different URLs and what is the information of those URLs

Input: Session id and the consecutive id.

Output: All the consecutives that are crawled, two sets of childs that are in the first level, common links in these sets and their information.

7) Given a session in HDE data, find out how a user can access unindexed webpages from indexed webpages. The query gets the indexed webpages that are crawled, from the HDE table and outputs the one level of unindexed childs of indexed web pages.

Reason: We want to find out how a user can access the unindexed webpages from indexed webpages.

Input: A session id

Output: The sets of lists of unindexed URLs that are connected to our indexed webpage in the first level.

8) Given a session in HDE data, find out the number of indexed and unindexed webpages.

Reason: This query is for a user to see a spesific session’s indexed and unindexed information.

Input: Session id is the input.

Output: The number of unindexed and indexed of selected session.

This list is an open one. We hope to learn how the unindexed web evolves using the above mentioned designs.

We are assuming that the queries fired will be done so by a field expert. Note that though the queries may return the necessary result, it is completely up to the user to draw the necessary conclusions.

Future Work

The initial work has so far involved the storage of url_crawler data and the linkage data. The linkage data as mentioned in the Fig 2 is stored as a digraph. The design of the Human-Driven events schema has been implemented and tested to completion. The Querying are mentioned as in the previous section.

In the future, more queries will be analyzed to see whether any more information can be retrieved and then tested whether they can be optimized. The storage of the data and the query firing will be optimized with minimal trade-offs of speed, storage and performance. There are ideas to index the database tables using the sequence id and the session id; specifically the Human-Driven Events Table, to make the querying more efficient.

Also it would be useful to check another property of URLs; whether a URL is *spam or not*. We can use spam-detection techniques for the same.

Almost all of the querying utilizes a Breadth First Search with a depth limit. Better ways of doing the same search will be analyzed since for every depth increase, the branching factor is exponential. A possible way to minimize this costly process is to prune the tree based on TLDs and depending on the query.

Additionally, we plan to integrate all the softwares in one package to facilitate the smooth running of all the software without any manual intervention. However, it must be noted that the codes for the initial testing and implementation are all mostly ***individual mutually-exclusive codes using command-line and HDEs*** *rather than a single framework running in tandem.*

Acknowledgements

The team would like to thank Dr. Yuqing 'Melanie' Wu and Dr. Minaxi Gupta for allowing us to work on their joint project. We appreciate Professor Wu's constant guidance, inputs and corrections.

We would also like to thank Youngsang Shin, Christopher Dunn and Mo Zhou for their valuable inputs, Crawler and Human-Driven Events generating tools and primarily, for their advice.

References

[1] The initial proposal paper 'NetSE: Large: A Foray into the UnWeb to Find Hidden Treasures and Hidden Maliciousness'

[2] Paper titled '*Web Mining and its SQL based Parallel Execution*' by Masaru Kitsuregawa, Takahiko Shintani and Iko Pramudiono: Institute of Industrial Science, University of Tokyo.

[3] Paper *'Breadth-First Search Crawling Yields High Quality Pages'* Marc Najork and Janet L. Weiner; WWW'01 Proceedings of the 10th international conference on World Wide.

[4] *'Graph Structure in the web'* Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, Janet Wiener.

[5] *'Duplicate Detection in Click Streams'* Ahmed Metwally, Divyakant Agrawal, Amr El Abbadi; Proceedings of the 14th International Conference on World Wide Web.

[6] *'Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data'* by Jaideep Srivastav, Robert Cooley, Mukund Deshpande, Pang-Ning Tan; Newsletter ACM SIGKDD Explorations Newsletter January 2000.

[7] *'A Web-Page Prediction Model Based on Click-Stream Tree Representation of User Behaviour'* by Sule Gunduz, M. Tamer Ozsu; Newsletter ACM SIGKDD'03 proceedings of the ninth ACM SIGKDD international conference on Knowledge Discovery and data mining.

[8] *`What did they do? Understanding clickstreams with the Web-Quilt visualization*

system' by Sarah J. Waterson, Jason I. Hong, Tim Sohn, James A. Landay, Jeffrey Heer, Tara Matthews; Proceedings of the Working conference on Advanced Visual Interfaces AVI'02.

[9] *'Introduction to the Formal Analysis of Social Networks using Mathematica'* by Luis R. Izquierdo, Robert A. Hanneman in Network (2006).