Collection Management Webpages

Final Report
December 25, 2017

CS 5604 Information Storage and Retrieval
Instructor: Edward A. Fox
Virginia Tech, Blacksburg, VA 24061

Submitted by

Eagan, Mackenzie	eaganm@vt.edu
Liang, Xiao	xliangvt@vt.edu
Michael, Louis	louism@vt.edu
Patil, Supritha	patil93@vt.edu

Collection Management Webpages

ABSTRACT

The Collection Management Webpages team is responsible for collecting, processing, and storing webpages from different sources. Our team worked on familiarizing ourselves with the necessary tools and data required to produce the specified output that was used by other teams in this class (Fall 2017 CS 5604). Input includes URLs generated by the Event Focused Crawler (EFC) [12], URLs obtained from tweets by the Collection Management Tweets team, and webpage content from Web Archive (WARC) files from the Internet Archive or other sources. Our team fetches raw HTML using the obtained URLs and extracts HTML from WARC files. From this raw data, we obtain metadata information about the corresponding webpage. The raw data is also cleaned and processed for other teams' consumption. This processing is accomplished using various Python libraries. The cleaned information is made available in a variety of formats, including tokens, stemmed or lemmatized text, and text tagged with parts of speech. Both the raw and processed webpage data are stored in HBase [19] and intermediately in HDFS (Hadoop Distributed File System).

Our team successfully executed all individual portions of our proposed process. We successfully ran the EFC and obtained URLs from these runs. Using these URLs, we created WARC files. We obtained the raw HTML, extracted metadata information from it, and cleaned and processed the webpage information before uploading it to HBase. We iteratively expanded on the functionality of our cleaning and processing scripts in order to provide more relevant information to other groups. We processed and cleaned information from WARC files provided by the instructor in a similar manner. We have acquired webpage data from URLs obtained by the Collection Management Tweets (CMT) team. At this time however, there is no end-to-end process in place.

Due to the volume of data our team has been dealing with, we explored various methods for parallelizing and speeding up our processes. Our team used the PySpark library for obtaining information from URLs and the multiprocessing library in Python for processing information stored in WARC files.

Contents

List of Figures

List of Tables

Chapter 1

Overview

1.1 Goals and Challenges

The Collection Management Webpages (CMW) team is responsible for collecting, processing, and storing webpages from different sources. These sources include URLs obtained using the Event Focused Crawler, tweets containing URL information from the CMT team, WARC files from the Internet Archive, and additional archives collected by Pranav Nakate, Mohamed Farag, and others.

For the first portion of the project, our time was spent mostly on developing the relevant background necessary for this project. We strove to develop an understanding of our duties as they relate to the other teams. As such, we spent the first few weeks of this project familiarizing ourselves with the related technologies and reading related literature.

To more effectively utilize our resources, our team chose to subdivide itself during this time. Each group had a specific focus and obtained a more in-depth knowledge about their focus. Our three groups were: running and understanding the Event Focused Crawler (commonly referred to as EFC), management and creating of WARC files, and storage of relevant information into HBase [12, 19].

Initially, the biggest challenge was on-boarding our entire team to this project. Since no one had an in-depth grasp on any of these topics, we all had to learn about what we will be doing; this reduced our output for the first report. This challenge is made greater by the fact that our team was missing a solid groundwork from last year's CMW team. Most of the work done to date by other teams, this year and last, was entirely focused on Twitter data as a result. While we do have resources available to use, like the EFC, many of the webpage related goals for last year were not achieved.

For the second part of this project, the biggest challenge was to define what our team was to produce in order to support the functionality of the other teams. Class discussions were

focused on defining what each team planned on definitively producing. Our team did not want to move forward with cleaning webpage data on a large scale until we had an accurate picture of what would be required of us. Information about what fields our team is supporting is discussed in detail in the Design portion of this document.

The other main goal for our team during this time period was to successfully obtain and write cleaned webpage text to the HBase database. We accomplished this goal. However, due to the discussions concerning the database schema, we did not move forward with populating the database with any other webpage data at that time. We investigated a variety of libraries available in the Python language. The Natural Language Tool Kit (NLTK) [18] library contains a variety of functionalities we have found to be useful for this task. We are also using the BeautifulSoup Python library to obtain and process webpage data.

The Tweet Framework, created by Matthew Bock as part of his Master's thesis [9], was investigated to understand if it would be possible for our team to use some of its functionality. Our team decided, due to time limitations, that we would be unable to modify his software to work with webpage data. The modification of this software may be a task for future CMW teams.

For the third part of this project, our team focused on expanding our Python script that contains all of the desired functionality in terms of cleaning and processing the webpage data. Our script now provides all types of information requested by the other teams. We also introduced a list of self-defined status codes that stores information regarding our attempts to fetch information from a given URL. The self-defined status codes are based in part on HTTP status codes and will be discussed in greater detail in the Design portion of this document.

One of the outputs produced by our team is WARC files generated from the collected URLs. To search through these files, a CDX file must first be created. We attempted to use a variety of tools to create the CDX files. We eventually found a Python tool, PYWB, that successfully created the CDX file [20]. While WARC files will not be used by the other teams in our class, they represent an important record of the data contained in the webpage at the time the information was retrieved. Preserving this information in the Internet Archive requires both the WARC file and the associated CDX file. We have also found them useful as a way of storing webpages from semester to semester as part of this course as we inherited numerous files related to school shooting events.

For the last portion of the semester, our team mostly focused on speeding up our current processes. Due to the volume of data that can be contained in a webpage, it can take several minutes to fetch the HTML information from a webpage and process it fully. Initially our team was looking at implementing some form of MapReduce [11] script to speed up this process. However, after discussions with the Graduate Teaching Assistant, Liuqing, we no longer believed this to be a viable option. At the time of writing, Python is not installed on all nodes of the class cluster, which meant that we could not run all of our code across different nodes. However, Liuqing informed us that Anaconda [5] has been distributed on

our cluster, and PySpark [8] could be used in the Anaconda environment. After investigating this Python Spark API, we have successfully parallelized the HTML fetching portion of our cleaning script. Due to time limitations and some issues with how WARC records are read in a Python script, we were unable to parallelize WARC record cleaning with PySpark; we have instead created a script that manages multiple processes to handle the different cleaning functions.

Lastly, we successfully obtained URLs from the CMT team. The information from these URLs has been fetched, cleaned, and stored into HBase. Our team has also run the EFC on certain solar eclipse tweet collections. Our team also processed all of the school shooting WARC files and placed that information into HBase.

1.2 Future Work

One of our goals this semester was to implement a pipeline from beginning to end. However, we were unable to accomplish this goal. We have all of the necessary functionality, but it does not flow smoothly from process to process and has introduced redundancy in fetching HTML, which currently happens multiple times. The EFC fetches to tell if a page is worth adding to its crawl and to obtain sub-URLs. Our Python script fetches, cleans, and stores every page it receives as a result of its URL list input. Lastly, when generating WARC files, we also fetch the HTML. Fetching is also a fairly slow process. A pipeline would make our project much simpler for users, as they would no longer need to remember to feed one output into the next step's input or to edit multiple configuration files.

Currently, we store a limited amount of information in our WARC files. We only store the HTML of a page. However webpages are rendered by most browsers using a combination of CSS (Cascading Style Sheet), HTML, and JavaScript. We would have liked to include CSS and JavaScript information in our created WARC files if possible. This would provide a more useful snapshot of what a webpage would look like at the time of fetching the information. While we did start working on WARC fairly early in the project, we wrapped up most of our work concentrating on this only at the end of the semester. This was because most of our team's attention was focused on the processes that were critical for other teams to proceed.

One of the ways our team receives input of URLs is from the CMT team, who provided expanded urls from tweets they collected. While we have obtained and processed URL information from the CMT team, our current processes for doing so may not be as efficient as they could be. We were provided with CSV files containing the tweet data from the CMT team. We then had to refine these to extract the tweets that we could use as inputs, that is, tweets containing URLs associated with them. From these URLs, we could only extract information from the webpages containing text information. We are unable to process other content such as images and videos. To filter the URLs to satisfy this condition, the crawler includes code to filter by MIME type. We use URLs that only have the MIME type as

'application/HTML'. Several tweets might be talking about the same article or webpage, hence such URLs would be repeated and might be crawled redundantly. To eliminate such superfluity, we extract only unique URLs and use only these as seeds for our crawler.

Lastly, though we have sped up our process using the PySpark and multiprocessing Python modules, further work is needed in this area. Processing still takes a large amount of time. The CMW team needs to work on increasing output of these processes for the betterment of the rest of the class. Precise suggestions are addressed in the following paragraphs.

In the cleaning process, one crucial step which can be further exploited by other teams is NER (Named Entity Recognition) [13]. The relevant code was produced by the Stanford Natural Language Processing Group and originally written in Java. Third-party libraries provided in other languages essentially call the Java commands and therefore reduce the performance. Our suggestion to future CMW team is to perform the cleaning step (at least the NER) in Java or Scala, whereas the crawling step can be performed in other languages such as Python which is used in EFC.

After exploring the PySpark framework on the cluster, we found it feasible for future CMW team to parallelize EFC using PySpark and a proper duplicate elimination design such as in the crawler structure mentioned in the course textbook [17]. We also suggest to record the running logs from the beginning of the class to better evaluate different designs and methods.

One area our team was interested in expanding further was to investigate the possibility of placing events into a queue upon the completion of our processing. Other teams may then be able to quickly pick up where our team left off with minimal downtime. The current way that we notify other teams of new data that has been processed is through directly communicating with them either in person, over Slack, or in email. Similarly, we are made aware of new tweet derived URLs via personal contact from the Collection Management Tweets (CMT) team. This can result in a non-negligible amount of downtime which could be significantly cut down if our scripts programmatically alerted each other to new information. We have discussed this issue with the CMT team, and we believe that having a column to indicate if a tweet derived URL has been explored will be a good solution.

Chapter 2

Literature Review

Chapters 19-21 in the course textbook [17] introduced the basic operation and architecture for a hypertext crawler. The scenario was originally set on web searching and indexing. The features discussed in these chapters can be applied to both small and large projects. This information includes more details about the distribution of a web crawler and refreshing previously crawled webpage copies. In this literature review, we will focus on the features more applicable to the size of the current project.

Chapter 19 discussed the heterogeneity of web content and the following difficulties of building a web search crawler. Web content is not only composed of text; it can also contain audio, images, and other rich media. Additionally, the content in webpages is usually not highly structured. Other challenges include link analysis and spam detection.

Chapter 20 discussed the features generally provided by a web crawler. The general requirements of a crawler are robustness and politeness. Robustness refers to making sure the crawler does not get stuck in one domain. Politeness refers to making sure the crawler is not burdening a server with too frequent requests. Lastly, a web crawler should return quality results. This chapter also described the basic architecture of a crawler.

Chapter 21 discussed how anchor text and extended anchor text can be used to determine if a given link is informative enough to be crawled. Language models are used to analyze the text. This chapter also discussed two main webpage scoring algorithms. The Event Focused Crawler we will be using employs a Vector Space Model algorithm to score webpages. This algorithm is similar in concept to the two scoring algorithms mentioned in this chapter.

During the course of the project, we have found Mohamed Farag's dissertation [12] and the report of the Collection Management Webpages team of the Fall 2016 semester [10] to be the most useful.

Farag's dissertation details the event focused crawler approach he implemented to obtain a collection of web URLs regarding a key event. As input, the crawler needs a diverse set of

high quality URLs to perform well. A resulting high quality webpage links to other relevant webpages and contains information about the event, such as the date and keywords. As such, our team will use other, crawled URLs besides the ones provided by the Collection Management Tweets team.

For the Event Focused Crawler to leverage the event model in predicting a webpage's relevance, Farag developed a function that measures the similarity between two event representations based on textual content. Additionally he implemented an additional source of evidence that allows the focused crawler to better estimate the importance of a webpage by considering its content. The webpage source importance is a dynamic value that changes as the crawl progresses. This value represents the the "number of relevant webpages that belong to a webpage source" [12]. The textual content information and source importance are combined into a single relevance score, which is then used to rank the links in order of relevance.

We read Helge Holzmann, Vinal Goel, and Avishek Anand's paper on "Archive Spark: Efficient Web Archive Access, Extraction and Derivation" to understand the role of Archive Spark in extracting and processing the files in WARC format [14]. The paper proposes Archive Spark, a framework for efficient, distributed Web archive processing to build a research corpus. While we initially planned on incorporating this information to process WARC files, we did not find enough documentation on Archive Spark to help us understand more about making use of this tool. So we decided to use the libraries available in Python to process WARC files.

Chapter 3

Requirements

3.1 Event Focused Crawler

The latest version of the EFC was provided by Liuqing; he also installed the software on the `efc2` virtual machine for our team [12].

Throughout the semester we monitored current world events and collect URLs relating to these events. At the time of writing, we collected URLs for the August 2017 solar eclipse, Hurricane Harvey, Hurricane Irma, and the 2017 Las Vegas Shooting using the EFC.

As the semester progressed, we altered the EFC in a few minor ways. We altered it to read information from a configuration file so that the code would not need to be directly altered. We also altered the code to remove hard-coded values for file names. Initially, our team had plans of configuring the EFC to integrate WARC file generation and altering the HTML fetching portion to be more efficient. Currently, the EFC fetches webpage text so it may obtain additional URLs present within the page. This process of fetching information from the World Wide Web can be fairly time consuming. We believe that fetching this data only one time would greatly improve efficiency.

3.2 Tweet Framework/Matthew's Framework

The framework in its current state is designed to process tweets to obtain the information contained within them [9]. The original plan was to alter the code of the framework so it could work with webpage data. Our team conducted further investigation of this software. As a result, we have determined it would not be feasible for our team to modify the software given the time constraint of a single semester. Our team shifted focus from using this framework to investigating Python libraries to clean webpage data.

3.3 HTML Fetching and HBase Ingestion

The main goal of our efforts was to deliver accessible information to the rest of the class such that they can complete their respective information processing tasks. Our method of distribution is to write raw and clean HTML, as well as some relevant metadata, to the corresponding columns in the HBase schema. The specific column details for this schema will be discussed in the Design section of this paper.

While we do have a script now in place to write all available, relevant information to HBase, this process can be made more efficient in future by automating the retrieval of HBase information, once the CMT team has written its resulting data there. Further, developing similar automation to streamline the process of running the EFC such that it automatically fills HBase with the raw and clean HTML from the pages it has crawled as well as the relevant metadata, would make the process more organized.

3.4 WARC Files

WARC files as input can come from two sources. The first source is the WARC file that will be generated from the list of URLs obtained from the EFC. The second source of WARC files is the Internet Archive or similar organizations.

When the Event Focused Crawler executes to generate a long list of URLs, we will run these URLs through a Python script that generates a WARC file and use another Python tool to create the CDX file. A CDX file is an index to WARC records. It contains the list of WARC records contained in the WARC file and an index to locate them. We use the CDX file to access metadata from any particular WARC record. We initially tried to do these two functions with the help of the 'wget' command. To avoid calling the command prompt from the script, we decided to use the warcio module of Python instead.

We were able to implement it successfully to convert the contents of webpages to a WARC file. Once we have a WARC file, we process it using the PYWB tool [20] to create the CDX file. After the creation of the WARC and CDX files, these files will be given to either Dr. Fox or Liuqing. Uploading the files to an entity such as the Internet Archive is out of the scope for our project. Our original goal was to collect at least one million documents from WARC files. We did not meet this goal, as our team severely underestimated the amount of time it would take to collect this quantity of data.

Initially, we had decided to use Archive Spark [14] to process data in WARC Files. Archive Spark is a framework that enables efficient data access, extraction, and derivation on Web archive data with a simple API for flexible and expressive queries. It is built on Apache Spark, which is an open-source cluster-computing framework and a general engine for large-scale data processing. However, our team decided to use an alternative tool due to the

limited amount of available documentation on Archive Spark.

We have instead chosen to implement a script using the BeautifulSoup, NLTK, and Stanford libraries to process the data. A similar file was used for collecting and processing information from webpages, and we chose a similar implementation to ensure uniformity. Once the required information has been extracted from the WARC files, the information will be ingested into HBase.

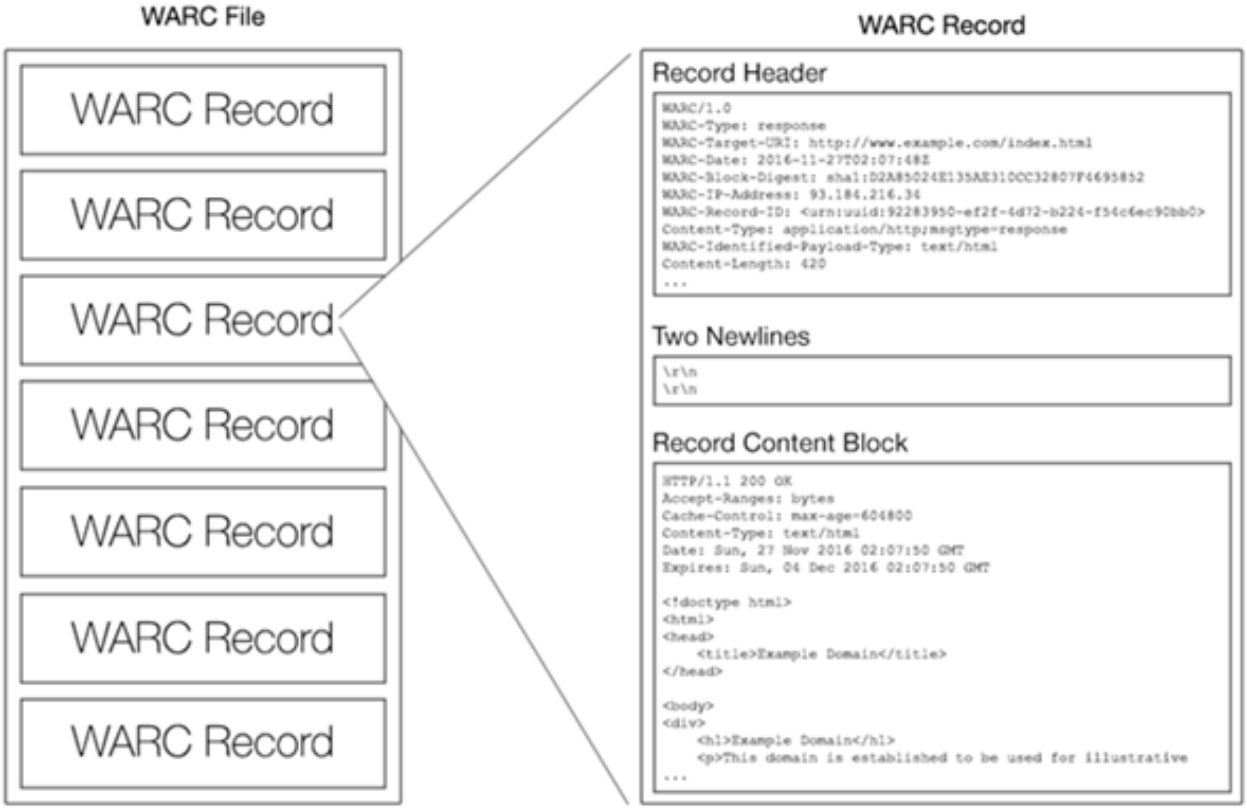

Figure 3.1: Representation of collection of WARC records in a larger WARC file [4]

Chapter 4

Design

4.1 Overview

One of our main goals for this project was to fully implement a workflow from start to finish; this workflow, as seen in Figure 4.1, will be discussed in the following section. We have obtained URLs from running the EFC, URLs from the CMT team, and WARC files from the instructor. We have successfully processed all forms of input. We store the required information in HBase according to a schema designed by the SOLR team. We use Python scripts to extract and clean webpage data.

Our team has developed all of the necessary scripts needed to complete our functions. However, for the most part, they are self-contained scripts and do not interact with one another.

4.2 Workflow

This section elaborates on the various processes in the workflow in Figure 4.1 as follows.

4.2.1 Focused Crawling

Description: The primary function of the Event Focused Crawler (using Mohamed's EFC source code [12]) is to generate a text file containing relevant URLs from the Web pertaining to a particular event with input as a list of user provided seed URLs.

Input: Model URLs, Seed URLs

Output: Relevant, focused crawled set of URLs

Tools Used: EFC source code, efc2 server (cs5604f17_cmw@efc2.cs.vt.edu)

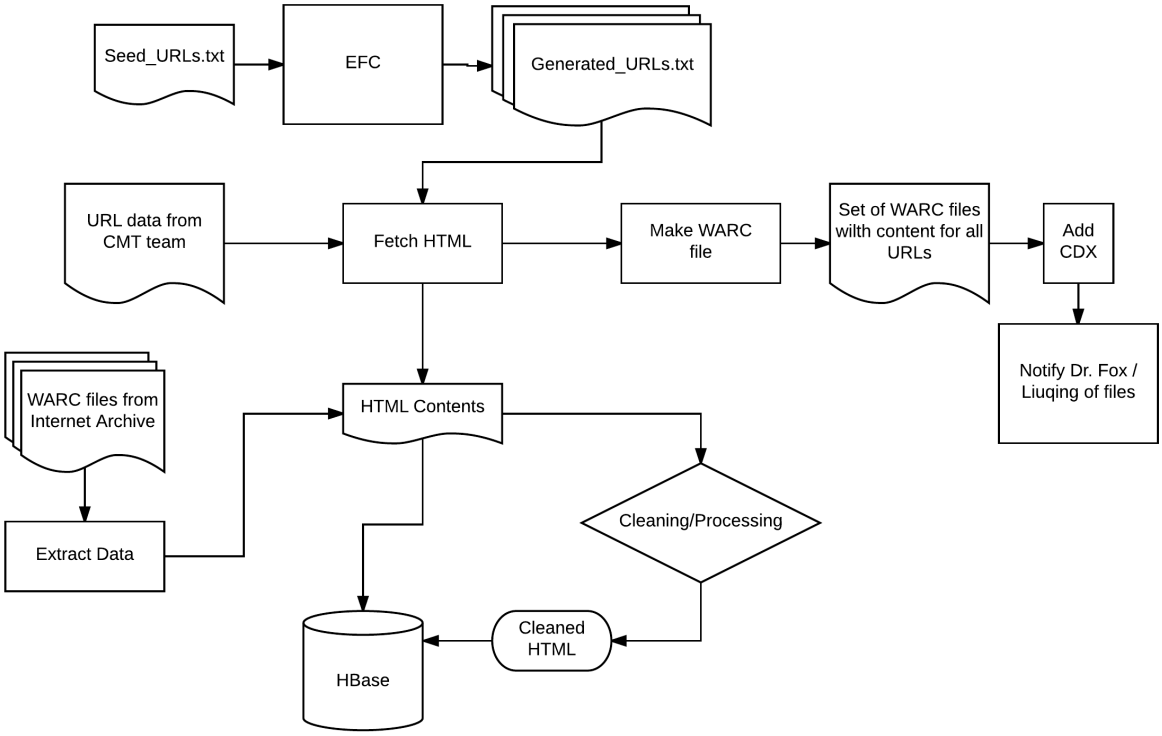

Figure 4.1: Current workflow of our project.

4.2.2 Cleaning/Processing

Description: Process the HTML content to fill the clean-webpage and webpage column families. Extraction involves parsing HTML trees, running Stanford Name Entity Recognition (SNER) on text, removing profanity, etc. In order to remove redundancy, we propose to do the cleaning and extraction process separately for URLs and WARC files. Since the team needed to fetch HTML documents from URLs and can simply use already stored HTML documents from WARC files. Cleaning and processing the data will require the use of various Python libraries and packages.

Input: Raw HTML data

Output: Cleaned text, tokenized text, stemmed text, lemmatized text, author/publisher, article publication date, etc.

Tools Used: Python NLTK library, Python scripts

Note regarding location: Over the lifespan of the project we explored several different ways to get location information from a webpage since latitude and longitude are required for the front end team to display documents on a map. Our initial approach was to simply record country specific domains; this is not a full-proof strategy since it is common to have

webpages that are not based out of the country specific domain they are associated with. An example of this is the common extension .gg, which corresponds to the crown dependency of Guernsey, but is frequently used for gaming websites due to the prevalence of the acronym GG for good game.

Instead of this approach we have chosen to collect name entity recognition information. This information is then passed on to the SOLR team. They, in turn, are planning to transfer this information into a latitude longitude pair with a corresponding bounding box. More information on the specifics of this process can be found in their report.

4.2.3 WARC Files

Processing WARC File
Input: Collection downloaded specific to an event (or topic, domain, etc.) from the Internet Archive or other source in the form of a WARC file (accompanied by CDX file) on HDFS

Output: TSV file webpage data for each WARC record in the WARC file (stored on HDFS). A Bash command provided by the SOLR team will be used to upload it to relevant columns in the "webpage", "clean-webpage" and "meta" column families in HBase.

Tools Used: Stanford NLP library, BeautifulSoup library, warcio library

Apart from extracting relevant columns in the "webpage" column family in the schema, we will also extract and sort rich information pertaining to a specific event using the Stanford NLP library.

Creating WARC Files
Input: List of URLs from the EFC or other source

Output: WARC file containing HTML for associated URLs (WARC files do not currently contain CSS or JavaScript) and associated CDX file

Tools Used: Warcio library, PYWB tool

4.3 Read and Write in HBase

The schema for HBase was generated by the SOLR team. The relevant section of the full schema is depicted in Table 4.1, Table 4.2, and Table 4.3. This is further broken down with more detail in Table 4.4, Table 4.5, and Table 4.6. The key of our HBase tables is <URL>-<Unix timestamp> for webpages we have fetched and <URL>-<ISO_DT> for information obtained from WARC files.

Table 4.1: Attributes in column family 'metadata' supported by our team.

Column-Name	Description	Example
doc-type	type of the document	tweet/webpage
collection-id	number of the collection	651
collection-name	name of the collection	electricity
dummy-data	designates if the data is dummy data or not	0/1

Table 4.2: Attributes in column family 'webpage' supported by our team.

Column-Name	Description	Example
html	raw HTML of webpage	raw HTML text
tweet-ids	unique identifiers of the tweets that contains the URL of this webpage	593392960886145024
language	webpage's main language	en
url	full URL of the webpage	http://www.roanoke.com/news
title	extracted title from the webpage	Student arrested after threatening Virginia Tech Yik Yak post
author/publisher	extracted author from the webpage	Tom LoBianco and Pamela Brown, CNN
sub-urls	URLs located in the content of the webpage	http://nytimes.com/story/id=5423
create-time	extracted create-time from the webpage	Mon Apr 13 19:00:21 +0000 2015
domain-name	extracted domain name from the webpage	http://www.fs.fed.us/
domain-location	extracted domain location from the webpage using meta tags	United States
organization-name	extracted copyright organization name in the page	Cable News Network
fetch-time	fetched time (readable)	Mon Apr 13 19:00:21 +0000 2015
mime-type	MIME type	image/jpg, image/png, text/html
status-code	self-defined status codes	0200, 0404, 0502, 1001, 2001

Table 4.3: Attributes in column family 'clean-webpage' supported by our team.

Column-Name	Description	Example
clean-text	clean text with HTML tags removed	Best US cities to watch 2017 total solar eclipse
clean-text-profanity	clean text with no profanity	This is {profanity} amazing
tokens	tokens from the clean-text-profanity column without stopwords	(event), (catastrophic)
stemmed	stems of words in clean-text	saw – s
lemmatized	lemmas of words present in clean-text	hurricanes – hurricane
real-world-events	a list of events in the webpage	Solar Eclipse; Flood
sner-people	extracted names of people in the page	Matthew, Alice, Bob
sner-organization	extracted organization names in the page	University of California, Museum of Natural History
sner-location	location that the event occurred, extracted from the article	California, United States
POS	the part of speech for all words in the documents	(hurricane, nn)
keywords	keywords of the webpage	solar eclipse

Table 4.4: This expands on the columns noted in Table 4.1 but adds specific information on how the CMW team is handling generating this information.

Column Name	CMW specific description
doc-type	In our case, the tag will always be "webpage" to be distinguished from the results of the CMT team
collection-id	If we fetched a document in association with an event focused crawl, we note specific collections with which these documents are associated.
collection-name	Human readable name associated with the collection-id

Table 4.5: This expands on the columns noted in Table 4.2 but adds specific information on how the CMW team is handling generating this information.

Column Name	**CMW specific description**
html	This is the raw HTML that we fetch using the URL; this will only be processed to remove newline characters and tabs in order to be written to a TSV.
tweet-ids	The corresponding tweet IDs for tweet-extracted URLs
language	The language of the page based on the text in the clean HTML
url	The fully defined URL for a webpage
title	The title of the page. This can be found in the metadata in the HTML if not left blank.
author/publisher	The author of the page. This can be found in the metadata in the HTML if not left blank.
create-time	The publication date of the article if it can be determined from the context of the page. Ways that this can be determined are from the URL of the page or the HTML of the page; further information on this technique can be found in [12].
sub-urls	Any webpage URLs that were linked to by the document that we are processing
domain-name	Domain name of the URL, naively derived from the URL
domain-location	Location of the country code naively derived from the domain name
organization-name	If the webpage has a certain copyright organization, the name of the organization shows here.
fetch-timestamp	The timestamp showing when the webpage is crawled and fetched.
mime-type	Read from header when information is fetched
status-code	The status code showing the response when the webpage is crawled and fetched. Status codes starting with "0" are HTTP status codes. Status codes starting with "1", "2" and "3" are our self-defined status codes.

Table 4.6: This expands on the columns noted in Table 4.3 but adds specific information on how the CMW team is handling generating this information.

Column Name	**CMW specific description**
clean-text	The raw HTML stripped of all of its HTML tags
clean-text-profanity	The clean text without profanity. Can be the output of either our team or the classification team.
tokens	List of tokens from the clean-text-profanity column created by using the Stanford Tokenizer present in the NLTK library [18], with stopwords removed. Tokens will include words and punctuation.
stemmed	Stems for all tokens in the clean text without profanity using the NLTK library.
lemmatize	Base forms of words for all tokens in the clean text without profanity using the NLTK library
real-world-events	Related events of the webpage. Some will be produced by the EFC or extracted from a collection of tweets. Otherwise this column will be left blank for other teams to fill in.
sner-people	Names of people mentioned in the article extracted using NLTK [18] interface of NER (Stanford Named Entity Recognizer) [13].
sner-organizations	Names of organizations mentioned in the article extracted using the NLTK interface of Stanford NER
sner-locations	Names of locations mentioned in the article extracted using the NLTK interface of Stanford NER
keywords	Keywords in the meta tag of the webpage

Chapter 5

Implementation

5.1 Timetable

Table 5.1: Denoting our time line of progress

Date	Schedule
8/31	Formed team. Established a Slack channel as primary means of communication
9/5	Facilitated large portions of the class joining the Slack channel. Set up meeting with Liuqing
9/6	Met with Liuqing to discuss the Event Focused Crawler and project as a whole
9/13	Successfully ran the Event Focused Crawler
9/20	Wget URL fetch processing developed
9/21	Interim Presentation 1
9/25	Python URL fetch processing developed
9/26	Pig script successfully executed, writing raw HTML to a test table
9/26	Interim Report 1
10/12	Wrote raw and cleaned HTML to HBase
10/17	Implemented WARC file generation
10/17	Interim Presentation 2
10/19	Interim Report 2
10/26	Started ingesting Internet Archive WARC files
10/31	Ingested new URLs as they were added to HBase for Hurricane Irma, the Las Vegas shooting, and the 2017 Solar Eclipse
11/2	Code reads for configuration file vs. requiring frequent updates
11/2	Discussion spawning the use of status codes and tracking MIME type
11/6	Updated our fetching and cleaning script to support running on different OS versions, writing status codes, and recording MIME type
11/7	Interim Presentation 3
11/9	Interim Report 3
11/22	Parallelized Processing of WARC files
11/28	Obtained Webpage data from URLs provided by CMT
11/29	Parallelized Processing of Webpage Data with PySpark
12/7	Final Class Presentation
12/10	Ran EFC code on URLs provided by CMT
12/11	Processed all School Shooting WARC files
12/14	Final Report Submitted
12/21	Revisions Implemented and Resubmitted to VTechWorks

Table 5.2: Completed work and purpose of that work

Functionality	**Purpose**	**Completed By**
Read EFC documentation	Familiarize ourselves with integral piece of technology for our group	All
Read Previous CS 5604 CMW Report	Familiarize ourselves with current progress and required work	All
WARC Creation	Preserve information on webpage at that specific date and time	Supritha and Mackenzie
Fetching and Cleaning Webpage Data	Obtain information from webpages for storage into HBase	Xiao and Louis
WARC Processing and Parallelization	Process over 100,000 records obtained from instructor provided WARC files	Mackenzie
Mime Types	Extract MIME type to ensure that only application/text are used to process	Xiao and Louis
Status Code	Define our list of status codes to keep records for URLs we processed	Xiao and Louis
Process Tweet URLs	Process tweets received from CMT team	Supritha
Investigate PySpark	Read documentation and perform small scale tests to see if tool is viable for our purposes	Xiao and Mackenzie
Webpage Parallelization	Convert original script to use PySpark to increase script efficiency	Xiao
EFC - self-collected URLs	Use EFC to obtain URLs related to the specified event; one of the main sources of input	Mackenzie
HBase	Use HBase in importing all data and managing interactions with other teams	Louis
Investigate MapReduce	See if using a MapReduce script is a viable option for the cluster	Louis
Investigate Matthew's Framework	See if the Twitter Framework could be altered to efficiently work with webpage data	Mackenzie

5.2 Best Methods

Through the course of this project we explored numerous different approaches for generating and processing data. The EFC [12] was chosen to crawl webpages due to its being well-tested, well documented, and proven to work well. While the current HTML fetching script does work, it runs too slowly to handle large amounts of data. We suggest future work for subsequent CMW teams: redesigning the script to work in a scalable manner. We chose to use the NLTK [18] library for a large portion of our processing tasks due to its versatility and its being a well used library. We decided to use BeautifulSoup [2] because it provides all of the functionality we needed, and is understood to be a standard for this style of processing.

We have also chosen to use the `urllib` Python module to avoid automatic system calls present when using the `wget` command. The PYWB [20] tool is used because of its ability to generate a CDX file. The PYWB tool also allows us to view the contents of a WARC file, which we found to be useful to ensure the file was created correctly.

Our team originally intended to use a MapReduce [11] script to help speed up our processing. However, after speaking to Liuqing, we learned that the necessary packages for this type of script might not be installed on all nodes of the cluster. Liuqing then suggested we look into the PySpark API [8], which he knew was distributed on the cluster in the Anaconda [5] environment. Currently, PySpark is used to parallelize the HTML fetching (crawling) step of our processing script.

Our team also tried to use PySpark to parallelize WARC file processing. We were unsuccessful. We suspect that the sheer amount of data being sent and how the data was formated did not match well with PySpark. Our team had written code before PySpark was successfully used that created multiple processes to handle the cleaning. The script was written in this way due to the familiarity one of the group members had with this type of processing. Our group decided to continue using this script due to time constraints. Further avenues should be explored to increase WARC file processing.

5.3 Evaluation

We performed several evaluations which led to some of the selections of our best methods. Total running time (crawling + cleaning) of different parallel methods using PySpark [8] is shown in Table 5.3. Percentage of reduction in total running time using the current best method (i.e., parallel crawling using PySpark) is shown in Table 5.4, total times of different runs are separated by comma. We can see from the results that with the help of parallel crawling, we are able to reduce the running time by approximately 25%. We have also tried to parallelize stemming using PySpark, which is a minor step in cleaning. The overhead was worse than our expectation and did not show a better performance. The total running time of parallel stemming is also depicted in Table 5.3. All the running times were measured on

the cluster. A baseline is given by the same processing steps without parallel computing.

We have also recorded the total running time as well as crawling time using parallel crawling given different URL sizes. The results are shown in Table 5.5. These records are missing a baseline to compare with, but to some extent show that the bottleneck of our performance is the unparalleled cleaning step.

Table 5.3: Total running time (in seconds) of different parallel methods using PySpark

URL Count	**Baseline (Unparalleled Processing)**	**Parallel Stemming**	**Parallel Crawling**
13	12.084	16.364, 21.165	15.496, 15.681, 20.150
1000	650.259	714.775	480.312,497.932

Table 5.4: Parallelized Crawling Evaluation Metrics, with time in seconds

Test Run	**URL Count**	**Baseline (Unparalleled Crawling)**	**Parallel Crawling**	**Reduction in Running Time**
Run 1	100	650.260	480.312	26.14%
Run 2	100	650.260	497.932	23.43%

Table 5.5: Total running time (in seconds) and crawling time (in seconds) using parallel crawling.

URL Count	**Total Running Time**	**Crawling Time**	**Percentage of Crawling Time**
992	1973.848	259.004	13.12%
2399	6055.408	2097.016	34.63%

Additionally we conducted comparisons for running times when processing and cleaning the information stored in WARC records. We do not have a direct comparison between cleaning data fetched from the Internet and data read from a WARC file. Before parallelization was introduced, we believed the WARC processing was more efficient, due to the fact it was reading from a file rather than sending out requests to the Internet. The parallelization here introduced a small amount of speedup as seen in Table 5.7. After moving the code to the cluster, we saw a significant speedup as shown in Table 5.8.

At the conclusion of the semester we placed 131,341 rows into HBase. These records came from a combination of all of the different input methods, discussed in Chapter 4, but the majority (111,020) of these records were from WARC files.

5.4 Collection Overview and Discussion

Over the course of the project we processed a large number of files broken down into several different collections. Each collection is a function of both how the team found the webpages processed, and the event these webpages are relent to. This is outlined in Table 5.6. While some of the webpages we processed extended existing collections since they were referenced in tweets provided by the CMT team we also propose the creation of several new collections to identify webpages processed from WARC files, generated by EFC runs and EFC runs directly, since these are distinct from collections related to tweets. The database and source of a collection dictate where the information for a collection originated while the ID distinguishes similar sourced collections from each other. The provided table also gives insight into the size of the contributions by CMW by noting the relevant TSV files the team generated and stored in HDFS and translated to HBase, as well as the number of rows, size in bytes and number of words in each of these files.

Database	**ID**	**Relevant Files in HDFS**	**Source**	**File Size in Bytes**	**Number of Rows**	**Number of Words**
Collect_yTK	997	August21 _finalURL _rst_bk.tsv	yTK	80023965	510	2877383
Collect_yTK	1005	oreclipse_ finalURL_rst _bk.tsv	yTK	198268054	992	7211720
Collect_yTK	1004	eclipseglasses_ finalURL _rst.tsv	yTK	314307651	2399	11391468
Collect_yTK	1006	totaleclipse_ finalURL_ rst.tsv	yTK	1291788259	8683	45438224
Collect_yTK	1003	totalsolar eclipse _finalURL _rst.tsv	yTK	468666168	2941	17731086
EFC	1035	cmwf17warc Dunbar Shooting Fixed.tsv	EFC	93293949	258	2664767
EFC	1036	cmwf17 warcNIU Fixed.tsv	EFC	3613081	19	200351

EFC	1043	cmwf17 warcUnivAlabama Shooting Fixed.tsv	EFC	43408661	328	1954949
EFC	1038	cmwf17 warc VTShootingFixed.tsv and cmwf17 warcVT Shooting OldFixed.tsv	EFC	548782342	3441	25783288
EFC	1042	cmwf17 TownvilleSchoolShooting.tsv	EFC	5507856620	25001	233282368
EFC	1041	cmwf17 UmpquaCollegeShooting.tsv	EFC	5271109819	25001	221934225
EFC	1039	cmwf17 WorthingSchoolShooting.tsv	EFC	5122981498	24996	242117318
EFC	41	cmwf17 sandyHook.tsv	EFC	1617477960	6459	84949401
EFC	1040	cmwf17 sparksMiddle.tsv	EFC	2701027338	25001	109765346
EFC	994	full.tsv	EFC	202313328	1344	10021291
EFC	1008	full.tsv	EFC	543987361	2726	24559829
EFC	1025	full.tsv	EFC	206921905	913	8592958

Table 5.6: Collections Contributed to and Proposed by the CMW Team

Table 5.7: WARC Cleaning - Local Computer (in minutes)

Event Associated with WARC File	**Number of Records**	**No Parallelization**	**With Parallelization**	**Percent Speedup**
Northern Illinois University Shooting	18	0.85	0.8	5.88%
Dunbar High School Shooting	258	11.83	10.74	9.21%
Reynolds High School Shooting	521	23.09	21.91	5.11%

Table 5.8: WARC Cleaning - Local Computer and Cluster (in minutes)

WARC File	**Number of Records**	**Parallelization - Local**	**Parallelization - Cluster**	**Percent Speedup**
Dunbar High School Shooting	258	10.74	6.68	37.8%
Reynolds High School Shooting	521	21.91	13.75	37.24%
VT University Shooting	1941	78.81	52.14	33.84%

5.5 Processing Time Breakdown

One of the main goals of any future work associated with this semester should be seeking to speedup the existing cleaning and fetching process. To better understand the current slowdowns a set of small scale tests were run to time the execution of each subprocess and are outlined in Table 5.9. Each test fetched a single URL and the time complete each step of the cleaning pipeline was recorded.

Table 5.9: Time Breakdown for Cleaning and Fetching Subprocesses

	Test 1	**Test 2**	**Test 3**	**Test 4**
Output File Size in Bytes	205496	180263	197303	332814
Number of Words	4121	10163	11499	16087
Time to Fetch HTML in seconds	0.421105862	0.68245101	0.808046818	0.308145046
Time to Parse HTML with BeautifulSoup in seconds	0.049103022	0.132313967	0.120690823	0.246493101
Time to Clean Text for Profanity in seconds	0.019669056	0.053640842	0.054510117	0.065136909
Time to Tokenize in seconds	0.000762939	0.012890816	0.013540983	0.018747091
Time to Label Parts of Speech in seconds	0.358403921	0.480870008	0.480540991	0.569386005
Time to Stem and Lemmatize in seconds	2.515690088	2.723646879	2.542860031	2.602529049
Time to Perform Name Entity Recognition in seconds	2.093142033	2.105861187	1.951417208	2.018919945
Time to Perform Language Estimate in seconds	0.650670052	0.787535906	0.793950081	0.789402008
Total Time in seconds	6.122077942	7.005435944	6.792712927	6.665222883

Chapter 6

User Manual

The most important output produced by our team is the information stored in the HBase database. The other teams will use the information in the database to perform their necessary functions. Our team throughout the semester strove to include features as requested by other teams. At the conclusion of the semester, we felt that we satisfied all of the needs we had been made aware of. Our team also creates WARC files. Although the WARC files will not be used by the other teams in this class, they will be given to the instructor for future use.

Our output information will be stored according to the schema designed by the SOLR team, which has been discussed in the design section of this paper. While the clean text information will contain more useful information with respect to the other teams' needs, the flags in place can allow other teams to know the current processing state for a given webpage.

In order to view or use any of the information currently in HBase, users can follow these steps to scan a table for data they are interested in:

Step 1: Set up the environment to access the cluster's HBase installation. This is most simply done by SSHing into the cluster and then optionally moving into node00 instead of node01. While on a VT network, use the command `ssh xxxx@hadoop.dlib.vt.edu` where xxxx is a valid user name for the cluster such as `cs5604f17_cmw`.

Step 2: Launch the HBase shell using the command `hbase shell`

Step 3: Scan the table you are interested in. Some commands to do this include `scan 'XXXXX'`, where XXXXX is a valid table such as `cmwf17-test` or `ideal-cs5604f17`. This outputs all of the information in a table and is not recommended; large tables will take an extended period to print to standard output. A more specific scan command is `scan 'cmwf17-test' , {COLUMNS => 'webpage:cleantext'}`; this only outputs data held in the column "cleantext" in the column family "webpage" in the table "cmwf17-test". More information can be found at [7].

This approach of scanning a table can be really useful at a glance but usually provides an overwhelming amount of information and is not very helpful for processing. To programmatically digest information in HBase it is recommended to use one of the APIs provided to work with HBase directly. A good getting started breakdown using Java can be found at [6].

The tables and the supported columns can be found in greater detail in the design section in Tables 4.1, 4.2, 4.3, 4.4, 4.5, and 4.6.

The CMT (Collection Management Tweets) team provided URLs that have been obtained from the collected tweets. This team also was responsible for expanding the URLs so that our team can obtain data from the long URLs. Our team learned late in the semester that Twitter shortens URLs using their t.co service. This could mean that URLs obtained from the CMT team will need to be expanded again if Twitter were to shorten an already short URL. An example of this is that the URL, "`https://t.co/NN0MeyA9m3`", is found in the raw text of a tweet and "`https://bitly.com/21AUG2017`" would be listed as an expanded URL, while both of these ultimately redirect to "`https://www.solar-eclipse.earth/`".

Chapter 7

Developer's Manual

This section provides an outline for any other groups that attempt to continue or modify our work. We think that the best resource to accomplish this task in conjuncture with this paper is the team GitHub located at `https://github.com/LouisMichael/cs5604cmw`. Forking this repository will give access to all of the programs and scripts discussed in this section.

7.1 Event Focused Crawler

The EFC requires having Python 2.7 installed. It also requires the following packages being installed: beautifulsoup4, bs4, certify, chardet, idna, ner, nltk, python-dateutil, pytz, requests, requests-cace, six, and urllib3. The EFC was developed as part of Dr. Farag's doctoral research [12].

Our team reviewed his work primarily through his dissertation.

The crawler was made more effective by creating and using a good model as well as seed URLs for webpages that are highly related to the event being crawled for. An ideal URL points to a webpage that contains date information and keywords related to the event. The EFC takes two text files, the model and seed files, that contain URLs. The same file can be used for both the model and seed URLs if the number of URLs to obtain is small enough. Variables can also be set that will affect how the EFC will run; the variables are located in an associated configuration file (`config.txt`). The variables can be altered without having to directly edit the code. One of the most important variables is the `pagesLimit` variable; this allows one to set the maximum number of URLs that will be crawled for. It is possible to crawl less than this number of webpages if the EFC runs out of unexplored webpages.

An example command to run the EFC is as follows. The resulting output is shown in Figure 7.1 which shows the count for the webpage crawled, the value the model determine

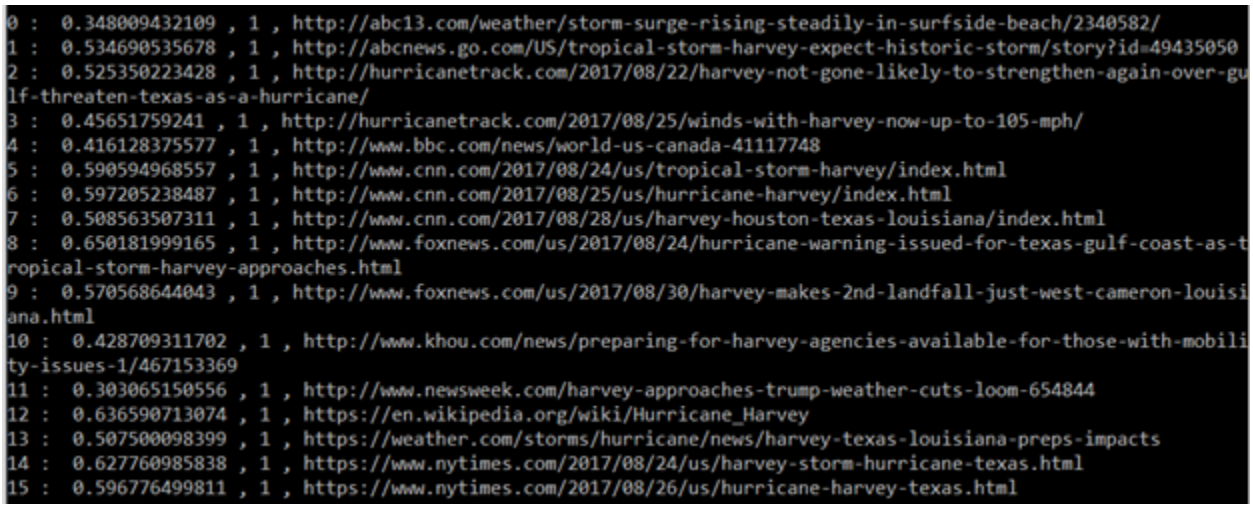

Figure 7.1: EFC output

for how strongly associated the webpage was to the event, and the URL for the crawled webpage.

```
python FocusedCrawler.py b input/model.txt input/seed.txt
```

Once the command has finished executing, either by running out of available links or hitting the page limit as defined in the `FocusedCrawler.py`, the found URLs will be written to a text file located in a sub-folder of the output folder. For the above example, the found URLs are located at `/EFC/output/input/model/base-webpages/base-Output.URLs.txt`.

7.2 WARC File Generation

For running the code to convert the HTML pages to WARC files, we use a Python script that runs on Python 3.6. The `warcio` and `warc` libraries in Python need to be installed to run the script. A WARC file will be created based around the event used in the EFC to obtain the URLs. The script will read the text file containing the URLs obtained by running the EFC.This script will then write all of the WARC records from a given EFC execution to a huge WARC files containing multiple WARC records, with a `warc.gz` file extension.

According to the ISO 28500 WARC file format specifications [1], 1 GB is the recommended size for a WARC file. It states that records over this size may become truncated. At this time we have not accounted for this issue, as the current WARC files that have been created have been small in size. Possible ways of dealing with this issue include monitoring the WARC file size or splitting the list of URLs into smaller subsets. This will need to be explored more in future projects.

To generate an accompanying CDX file [3] for each of the WARC files, the Python PYWB

tool [20] was used. Instructions for installing this tool were found on the tool's GitHub page. This tool requires a variety of packages to be installed; the packages and their version information can be found in Table 7.1. These packages are the dependencies that the team leveraged in our code base and would need to be installed by any team hoping to run our code in the future. We ran the start command, `wayback`, to receive print outs of error messages that informed us of what packages were missing. Once installed, the command `cdx-indexer --sort <output directory or file> <input directory or file>` can be executed to create a CDX file.

Table 7.1: PYWB Required Packages

Module/Distribution	**Version**
asn1crypto	$\geq$ 0.21.0
brotlipy	-
certauth	$\geq$ 1.2
certifi	$\geq$ 2017.4.17
cffi	$\geq$ 1.0.0
chardet	-
cryptography	$\geq$ 1.9
enum34	$\geq$ 1.0.4
gevent	= 1.2.2
greenlet	$\geq$ 0.4.10
idna	$\geq$ 2.5
ipaddress	-
jinja2	= 2.8
MarkupSafe	-
portalocker	-
pycparser	-
pyopenssl	-
pywb	= 0.52.0
pyyaml	-
redis	-
requests	-
requests-file	$\geq$ 1.4
six	-
surt	$\geq$ 0.3.0
tldextract	$\geq$ 2.0
urllib3	$\geq$ 1.21.1
warcio	$\geq$ 1.5.0
webassets	$\geq$ 0.21.1
webencodings	-
werkzeng	-
wsgiprox	$\geq$ 1.4.1

The PYWB tool also allows one to view the contents of the WARC file using the CDX file. The `config.yaml` file must be edited to point to the correct paths for the index and WARC files. The server then needs to be started using the `wayback` command. Using this

functionality, our team was able to test that both the WARC file and the CDX file were created correctly.

7.2.1 Future Work

Our WARC records only contain the HTML from a webpage. CSS and JavaScript information is not included. Our team would like to improve upon the information being retrieved and stored in the WARC files by including accompanying CSS JavaScript for webpages.

7.3 HTML Fetching

The first implementation of the code to fetch URLs can be found at our team's repository[1] in the `crawler_cleantxt_advanced.py` file. This Python script uses the Python library `requests` [15] to fetch webpage contents and `bs4` [2] to parse and extract basic information from HTML.

This script uses the Python library `nltk` [18] to perform natural language processing such as tokenization, stopword removal, stemming, and lemmatization. It also uses `nltk` to output the corresponding information such as POS. For NER (Named Entity Recognition), this script uses the interface provided by `nltk` to the Stanford NER [13] written in Java. To correctly perform NER, developers need to download the Stanford Named Entity Recognizer `jar` file from the The Stanford Natural Language Processing Group website [2] and put it under the correct directory. The current default directory for the NER models is `stanfordner/classifiers/`.

Currently, the result of fetching the HTML is stored in a TSV file after it is stripped of all of its tab, newline, and return characters.

The next version of our implementation handles situations when we failed to fetch a webpage or the webpage content is of a non-text type. For each of the URLs we process, we obtained and recorded the MIME (Multipurpose Internet Mail Extensions) type of the content by extracting the Content-Type header of the webpage using the Python library `requests`. We also obtained and recorded the HTTP status code every time when we attempted to fetch a URL. We only proceed with information extraction if the HTTP status code is "200" and the MIME type of the webpage is "text/html". In other cases we do not process the information fully, and we record our self-defined status code as in Table 7.3 to note this occurrence.

This approach led to issues with performance, especially when processing the large collection of tweets we obtained from the CMT team. As mentioned in Chapter 5, we finalized our

[1] https://github.com/LouisMichael/cs5604cmw/
[2] https://nlp.stanford.edu/software/CRF-NER.shtml

implementation with parallel crawling using PySpark [8]. The implementation of the code can be found at our team's repository along with a README file on how to configure and run the scripts. The `crawler_cleantxt_crawl_spark.py` file is for a URL list input while `crawler_cleantxt_crawl_spark_twitter_list.py` file is for a collection of tweets input. For each input, we read in all the URLs as a list. For the URL list we perform duplicate elimination (precisely in our case we turn the list to a set), then use the unique URLs as the key to parallelize the crawling using PySpark. The lists of webpage content are then collected as the returned values. Each time we parallelize our crawling on a predefined URL size (in our case we used 1000). After collecting all the webpage contents for this number of URLs, we perform the cleaning step and output the extracted information to a TSV file. Then if there are more URLs to crawl, we go back to the parallel crawling step and repeat the process.

Table 7.2 shows the number of rows of the original collection of tweets, tweet records with sub-URLs, and tweet records with unique URLs. The number of rows dropped amazingly after we performed uniqueness checking, especially for a relatively large collection of tweets. Table 7.4 shows some metrics on some of our result output TSV files. TSV files starting with "cmwf17" are outputs from EFC; other files are outputs from the collection of tweets in HBase. The output files from EFC were crawled before we implemented our self-defined status codes, so the number of fetched URLs is the same as the total URLs. The average number of words in each column shown in the table is calculated using valid (i.e., non-NA) rows only. See Appendix A for an example of the columns and corresponding values we have in HBase.

Although the data size is relatively small to allow a complete analysis, we can get some interesting insights from the metrics, such as there are more people (longer average number of words in "sner-people" column) mentioned in tweets than webpages and there are more locations (longer average number of words in "sner-location" column) mentioned in the solar eclipse event collection than for the Vegas shooting event.

Table 7.2: Metrics on collections of tweets

Collection of tweets	**# total rows from CMT**	**# rows with sub-URLs**	**# rows of unique URLs**
August21	2017	690	510
oreclipse	15243	2411	992
eclipseglasses	20618	3364	2399
totalsolareclipse	26788	4703	2941
totaleclipse	1005938	41735	8683

At the conclusion of the semester we were aware of some problems in our current implementation on extracting NER features. The NER output was in the form of separated words instead of phrases. Stanford Language Processing Group offers an official library in Java

and lists several third-party libraries in Python as well as other languages[3] that could be explored for solutions to this issue.

Once the NER location phrases can be correctly recognized, it becomes feasible to map documents onto real world locations [16]. This process was being conducted by the SOLR team and is discussed more in their report and presentation.

Table 7.3: Our self-defined status codes

Status code	**Situation**
0XXX	"0" + HTTP status code we obtained; see the list of HTTP status codes for details
1001	Content-Type is not text/html
2000	Content-type is text/html, but no text data extracted (rare case)
2001	Failed to fetch the webpage due to unknown reason. No HTTP status code returned
3200	Record was obtained from a WARC file

We have also made our script adaptable to different operating systems (Linux, MacOS, Windows) as well as Python versions (Python 2.x and Python 3.x). The operating system and Python version are detected at runtime.

The change from Bash and Scala framework to Python was based on team experience and the overall belief that it would have a smaller learning curve for most of the cleaning and processing that we are trying to accomplish. We also wanted to integrate our current implementation on cleaning and parallelizing with EFC, which was originally written in Python.

[3]https://nlp.stanford.edu/software/sutime.html

7.4 WARC Processing

The original WARC file cleaning script took the original `crawler_cleantxt_advanced.py` script and modified it to work with WARC files. The main modifications were to reading file input and obtaining the HTML. The warcio Python library was used to read information from the WARC files. To parallelize the processing, our team created a script that uses the multiprocessing Python module to create multiple processes, one for most major functions, and pipes to communicate between these processes. This leverages multiple cores on a single node of the cluster. According to the documentation, the multiprocessing module allows the program to use multiple processors on the machine that executes the program. A pipe has two 'ends', which can both be used for reading and writing. Reading is a blocking call; if there is nothing in the pipe when a process attempts to read from it, the process will wait there until there is something to read.

```
def getPOS(pipe):
    allTokens = pipe.recv()
    while allTokens != -1:
        POS = nltk.pos_tag(allTokens)
        try:
            parts = ['{}:{}'.format(i, j) for (i, j) in POS]
        except:
            parts = ['']
        pipe.send(parts)
        allTokens = pipe.recv()
```

Figure 7.2: Sample of code from the `warcParallel.py` script

The main difference when utilizing processes is that the script creates all needed processes and pipes at the beginning of the program's execution. The pipe for an associated process is passed as a parameter to the process's related function. After all records from the WARC file have been processed, the processes and pipes must be closed. To do this, the main process sends a 'kill' signal in the form of a -1. As shown in Figure 7.2, the process will initially wait for the main process to send information to it. Once the information has been sent, it will begin to be processed. Once processing is complete, it will send the information back to the main process and then wait again for the main process. The main process, in turn, has a read call for this information, though the main process is structured in such a way to reduce waiting.

The `cleanParallel.py` script was first tested on one group member's local machine. This machine was running on a x64-based processor with an Intel 7th generation i7 processor and 8 GB of RAM. It was tested against the original, non-parallelized script to see if there was any improvement in running time. As shown in Table 5.7, there was a small speedup of less

than 10% for the three listed records. While two files have similar speedup values, the outlier value of 9.21% is likely caused by other processes on the local machine taking up resources.

The script was then altered in a few minor ways so that it could be run on the cluster, to reduce dependencies and integrate PySpark which runs our code across multiple nodes on in the cluster. The general python executable cannot be used to run the `cleanParallel_cluster.py` script, as it is missing necessary packages. The Python executable located at `/opt/cloudera/parcels/Anaconda/bin/python` is the one that should be used. The warcio package may also need to be downloaded from the warcio GitHub page `https://github.com/webrecorder/warcio` and stored in the same directory as the Python script. The code was then run on the cluster to compare times against those from the personal laptop. As shown in Table 5.8, there was an impressive speedup of over 30% for the three files shown.

Table 7.5 shows running times for all WARC files and what platform the script was run on. The time is missing for the Sandy Hook School shooting file due to an unknown error that occurred while running the cleaning script.

We were unable to locate the source of this error. However, this table clearly shows it takes a large amount of time to process roughly 111,000 records. Moving forward, finding other methods for decreasing the amount of time spent will be invaluable.

Table 7.4: Metrics of result TSV files

File name and event	**# total rows**	**# rows with status code 0200**	**# rows with NER**	**Average number of words in column**
August21_rst.tsv event: solar eclipse	510	470	location: 254, people: 174, organization: 76	location: 9.66, people: 10.66, organization: 7.43, tokens: 531.20
cmwf17solar.tsv event: solar eclipse	755	755	location: 169, people: 115, organization: 51	location: 6.58, people: 3.07, organization: 4.75, tokens: 839.56
cmwf17vegas.tsv event: Las Vegas shooting	912	912	location: 115, people: 508, organization: 199	location: 3.06, people: 4.67, organization: 9.39, tokens: 1257.29
eclipseglasses_rst.tsv event: solar eclipse	2399	2261	location: 879, people: 603, organization: 203	location: 7.57, people: 11.12, organization: 8.28, tokens: 378.43
oreclipse_rst.tsv event: solar eclipse	992	919	location: 667, people: 409, organization: 144	location: 7.87, people: 13.05, organization: 8.05, tokens: 712.11
totalsolareclipse_rst.tsv event: solar eclipse	2941	2754	location: 1521, people: 1187, organization: 461	location: 11.16, people: 10.37, organization: 8.85, tokens: 538.87

Table 7.5: WARC Cleaning - Overall Time (in minutes)

WARC File	**Number of Records**	**Parallelized Time**	**Location**
Dunbar High School Shooting	258	6.68	Cluster
North Illinois University Shooting	18	0.8	Local Machine
Reynolds High School Shooting	521	13.75	Cluster
Sandy Hook Elementary Shooting	6459	n/a	Cluster
Cluster Sparks Middle School Shooting	25000	612.03	Cluster
Townville Elementary School Shooting	25000	733.20	Cluster
Umpqua Community College Shooting	25000	670.72	Cluster
University of Alabama Shooting	328	13.67	Local Machine
VT Shooting	1941	52.14	Cluster
VT Shooting (Old)	1500	61.38	Local Machine
Worthing High School Shooting	24995	705.44	Cluster
	Total Number of Records: 111,020	Total Time Spent: 2,869.81 minutes = 47.8 hours	

Chapter 8

Acknowledgements

Our team would like to thank Dr. Fox for his guidance during this project. We would also like to thank Liuqing Li for answering our questions and helping us to get up and running. Our efforts contribute and benefit from the work done on the Global Event and Trend Archive Research project funded by NSF Grant IIS-1619028.

Chapter 9

Bibliography

[1] Information and Documentation - the WARC File Format, 2008. URL: http://archive-access.sourceforge.net/warc/WARC_ISO_28500_final_draft%20v018%20Zentveld%20080618.doc.

[2] Beautiful Soup Documentation, 2015. Last accessed 10/19/2017. URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

[3] The CDX File Format (2015), 2015. Last accessed 12/14/2017. URL: https://iipc.github.io/warc-specifications/specifications/cdx-format/cdx-2015/.

[4] WARC File and WARC Record Example Illustration., 2015. Last accessed 11/08/2017. URL: https://www.mixnode.com/docs/reading-your-data/the-warc-format-explained.

[5] Anaconda: Python data science platform, 2017. Last accessed 12/12/2017. URL: https://www.anaconda.com/.

[6] Example: HBase APIs for Java "Hello World" Application, 2017. Last accessed 10/13/2017. URL: https://cloud.google.com/bigtable/docs/samples-java-hello.

[7] HBase - Scan, 2017. Last accessed 10/15/2017. URL: https://www.tutorialspoint.com/hbase/hbase_scan.htm.

[8] Python programming guide, 2017. Last accessed 12/12/2017. URL: https://spark.apache.org/docs/0.9.0/python-programming-guide.html.

[9] Matthew Bock. A framework for Hadoop based digital libraries of tweets. Master's thesis, Virginia Tech, 2017. URL: http://hdl.handle.net/10919/78351.

[10] Tung Dao, Christopher Wakeley, and Liu Weigang. Collection Management Webpages - Fall 2016 (CS5604). 2017. URL: https://vtechworks.lib.vt.edu/handle/10919/76675.

[11] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008. DOI: 10.1145/1327452.1327492. URL: https://dl.acm.org/citation.cfm?id=1327492.

[12] Mohamed Magdy Gharib Farag. *Intelligent event focused crawling*. PhD thesis, Virginia Tech, 2016. URL: http://hdl.handle.net/10919/73035.

[13] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proceedings of the 43rd annual meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics, 2005. URL: http://nlp.stanford.edu/~manning/papers/gibbscrf3.pdf.

[14] Helge Holzmann, Vinay Goel, and Avishek Anand. ArchiveSpark: efficient web archive access, extraction and derivation. In *Digital Libraries (JCDL), 2016 IEEE/ACM Joint Conference on*, pages 83–92. IEEE, 2016. URL: http://ieeexplore.ieee.org/abstract/document/7559568/.

[15] Kenneth Reitz. Requests: HTTP for Humans, 2017. Last accessed 11/08/2017. URL: http://docs.python-requests.org/en/master/.

[16] Geocoding library for Python. A Python 2 and 3 client for several popular geocoding web services., 2017. Last accessed 12/12/2017. URL: https://github.com/geopy/geopy.

[17] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. Introduction to information retrieval. 2011. URL: https://nlp.stanford.edu/IR-book/.

[18] NLTK Project. Natural language toolkit, 2017. Last accessed 11/08/2017. URL: http://www.nltk.org/.

[19] Apache Software Foundation. HBase. 2017. URL: http://hbase.apache.org.

[20] Python WayBack Tool Team. Python WayBack for web archive replay and live web proxy, 2017. Last accessed 11/08/2017. URL: https://pypi.python.org/pypi/pywb/0.33.2.

Appendix A

Detailed Breakdown of One HBase Row

This appendix goes over one full row in HBase to provide an example of a processed document. For this example we chose to review the contents generated from the URL `http://ow.ly/RFKv30ezOLV`. Each subsection of this appendix provides the contents of a column in HBase.

Since many of the values in this example such as webpage:html are infeasible to include verbatim in this appendix a TSV example is also provided in the supporting documents of this report as `appendixAExample.tsv` or can be found at `https://drive.google.com/open?id=1AhiWM0DvuXKSjQxGt-2hiuVROzUvLSvQ`. Truncated rows are listed in Table A.1

Table A.1: Truncated Rows in A.2

webpage:html
webpage:sub-urls
clean-webpage:clean-text
clean-webpage:clean-text-profanity
clean-webpage:tokens
clean-webpage:stemmend
clean-webpage:lemmatized
clean-webpage:POS
lean-webpage:sner-people
lean-webpage:sner-organizations
clean-webpage:sner-locations

Table A.2: Example Row

Key	http://ow.ly/RFKv30ezOLV-1512519174.66
metadata:doc-type	webpage
metadata:collection-id	1003
metadata:collection-name	#totalsolareclipse
webpage:url	http://ow.ly/RFKv30ezOLV
webpage:html	<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd"> <html lang="en"><!--[if lt IE 7]><html lang="en" class="ie6"><![endif]--><!--[if IE 7]><html lang="en" class="ie7"> <![endif]--><!--[if IE 8]> <html lang="en" class="ie8"><![endif]--><!--[if IE 9]><html lang="en" class="ie9"> <![endif]--> <!--[if (gt IE 9)|!(IE)]><html lang="en"><![endif]--><head><META http-equiv="Content-Type" content="text/html; charset=utf-8"><!--Customized Stylesheet Includes - prehead--><!--Customized JS Includes - prehead--><title>RFD-TV Team Enjoys Total Eclipse</title>..<meta name="Description" content="See photos as the team took in the phenomenon from the roof.
webpage:language	en
webpage:title	RFD-TV Team Enjoys Total Eclipse
webpage:author/publisher	
webpage:organization-name	
webpage:create-time	
webpage:domain-name	http://ow.ly/
webpage:domain-location	ly
webpage:sub-urls	http://www.rfdtv.com/story/36143195/are-you-rea
webpage:fetch-time	1512519174.66
webpage:mime-type	text/html; charset=utf-8
webpage:status-code	0200
webpage:tweet_ids	900022913922760704,900000102705758209, 899993369987952640,899993339113742337

clean-webpage:clean-text	FD-TV Team Enjoys Total Eclipse RFD-TV Team Enjoys Total Eclipse.RFD-TV Team Enjoys Total Eclipse.....RFD-TV Team Enjoys Total Eclipse.SITE SEARCH.WEB SEARCH BY .....NEWS.WESTERN SPORTS.WEATHER.THE PLAYBACK.SHOWS.Agriculture.Equine .Music & Entertainment .Rural Lifestyle .RURAL AMERICA LIVE.Western Sports.SCHEDULE.EVENTS.FIND US.WATCH RFD-TV...Print.....RFD-TV Team Enjoys Total Eclipse.....
clean-webpage:clean-text-profanity	RFD-TV Team Enjoys Total Eclipse RFD-TV Team Enjoys Total Eclipse.RFD-TV Team Enjoys Total Eclipse.....RFD-TV Team Enjoys Total Eclipse.SITE SEARCH.WEB SEARCH BY .....NEWS.WESTERN SPORTS.WEATHER.THE PLAYBACK.SHOWS.Agriculture.Equine .Music & Entertainment .Rural Lifestyle .RURAL AMERICA LIVE.Western Sports.SCHEDULE.EVENTS.FIND US.WATCH RFD-TV...Print.....RFD-TV Team Enjoys Total Eclipse.....
clean-webpage:keywords	
clean-webpage:tokens	fd,tv,team,enjoys,total,eclipse,rfd,tv,team, enjoys,total,eclipse,rfd,tv,team,enjoys,total, eclipse,rfd,tv,team,enjoys,total,eclipse,site, search,web,search,news,western,sports,weather
clean-webpage:stemmend	rfd,tv,team,enjoy, total,eclips,rfd,tv,team,enjoy, total,eclips,rfd,tv,team,enjoy, total,eclips,rfd,tv,team,enjoy, total,eclips,site,search
clean-webpage:lemmatized	rfd,tv,team,enjoys,total,eclipse,rfd,tv,team,enjoys ,total,eclipse,rfd,tv,team,enjoys,total,eclipse,
clean-webpage:POS	rfd:JJ,tv:NN,team:NN,enjoys:VBZ,total:JJ,eclipse:NN, rfd:NN,tv:NN,team:NN
clean-webpage:sner-people	tim,ross,mike,sacci,clifton,larson,allen, clifton,larson,allen,katelyn,mccullock,katelyn,
clean-webpage:sner-organizations	rural,health,national,rural,health,association, national,rural,health,association,house
clean-webpage:sner-locations	america,sydney,australia,america,america, alaska,alaska,america,america,
clean-webpage:real-world-events	totalsolareclipse