Collection Management Webpages

Final Report

December 8, 2016

CS5604 Information Storage and Retrieval

Virginia Tech

Blacksburg, Virginia

Fall 2016

Submitted by

Dao, Tung	tungdm@vt.edu
Wakeley, Christopher	chrisiw@vt.edu
Weigang, Liu	qfsdy@vt.edu

Instructor

Prof. Edward A. Fox

Abstract

The Collection Management Webpages (CMW) team is responsible for collecting, processing and storing webpages from different sources including tweets from multiple collections and contributors, such as those related to events and trends studied in local projects like IDEAL/GETAR, and webpage archives collected by Pranav Nakate, Mohamed Farag, and others. Thus, based on webpage sources, we divide our work into the three following deliverable and manageable tasks. The first task is to fetch the webpages mentioned in the tweets that are collected by the Collection Management Tweets (CMT) team. Those webpages are then stored in WARC files, processed, and loaded into HBase. The second task is to run focused crawls for all of the events mentioned in IDEAL/GETAR to collect relevant webpages. And similar to the first task, we would then store the webpages into WARC files, process them, and load them into HBase.

We also plan to achieve the third task which is similar to the first two, except that the webpages are from archives collected by the people previously involved in the project. Since these tasks are time-consuming and sensitive to real-time processing requirements, it is essential that our approach be incremental, meaning that webpages need to be incrementally collected, processed, and stored to HBase. We have conducted multiple experiments for the first, second, and third tasks, on our local machines as well as the cluster. For the second task, we manually collected a number of seed URLs of events, namely “South China Sea Disputes”, “USA President Election 2016”, and “South Korean President Protest”, to train the focused event crawler, and then ran the trained model on a small number of URLs that are randomly generated as well as manually collected. Encouragingly, these experiments ran successfully; however, we still have to work to scale up the experimenting data to be systematically run on the cluster. The two main components to be further improved and tested are the HBase data connector and handler, and the focused event crawler.

While focusing on our own tasks, the CMW team works closely with other teams whose inputs and outputs depend on our team. For example, the front-end (FE) team might use our results for their front-end content. We discussed with the Classification (CLA) team to have some agreements on filtering and noise reducing tasks. Also, we made sure that we would get the right format URLs from the Collection Management Tweets (CMT) team. In addition, the other two teams, Clustering and Topic Analysis (CTA) and SOLR, will use our team’s outputs for topic analyzing and indexing, respectively. For instance, based on the SOLR team’s requests and consensus, we have finalized a schema (i.e., specific fields of information) for a webpage to be collected and stored.

In this final report, we report our CMW team’s overall results and progress. Essentially, this report is a revised version of our three interim reports based on Dr. Fox’s and peer-reviewers’ comments. Besides to this revising, we continue reporting our ongoing work, challenges, processes, evaluations, and plans.

Table of Contents

Table of Figures

Table of Tables

1. Overview

In the previous section, we set out our team's three main tasks that we would like to achieve incrementally during the semester. The first thing we prioritized is learning and understanding the techniques and tools for working with URLs, webpages and WARC files, because none of us had any relevant background. Secondly, we started to learn and familiarize ourselves with new related and required concepts and technologies, such as the HDFS file system [7], HBase database [4], Hadoop [8], and web crawling and processing [9]. Tools that we have investigated include, Heritrix and Nutch (i.e., open-source Java-based tools for crawling and archiving webpages), Apache Pig (for saving and loading big data to HBase), and warcbase for managing web archives on HBase.

In addition to researching and learning the essential relevant background, and cutting-edge technologies, we also studied reports by students in previous semesters of the course. Especially, we found Mohamed Farag's dissertation [1] very useful in understanding the concepts and technologies for event focused crawling. Also, previous reports in the past related to noise reduction and Named Entity Recognition (NER) helped us build a basic understanding of designing and coding our system. From the very beginning of the class, we have started building the system incrementally by experimenting with a small data file that was assigned to our group in the Hadoop cluster. For example, we could use JSoup [11] and MySQL [14] to build a simple web crawler in Java, running successfully on a local machine. Such an example can be found here [15]. In the first report, we stated that we would later plan to incrementally scale it up to work on clusters (i.e., IDEAL/GETAR's servers) [10]. In the second report, after multiple emails exchanged with Mohamed Farag, we learned that we could reuse Mohamed Farag's focused crawling engine (possibly with some modification). Therefore, we decided to follow that direction because his focused crawler is well designed and tested, saving us plenty of time and effort. Nevertheless, due to the fact that Mohamed was refactoring his source code, we did not have a chance to use it, and report its operation in the second report. Fortunately, the crawler source code was finally handed to us a few days after the second report was submitted, and we were able to run it on the efc2 server. Since then we have worked hard to have the focused crawler run successfully with some small sample of data, at this point, on a local machine. Due to some technical issues with the server's privileges, we weren't able to run the crawler on the DLRL cluster initially. Fortunately, with Islam's help, we fixed this issue and ran the crawler successfully on efc2. Besides the webpage sources collected by the focused crawler, we also considered other sources of webpages that were already collected and classified; one of which is a cleaned and classified webpage archive about school shooting events collected by Pranav Nakate in his independent study. Unfortunately, after contacting Pranav and Mohamed, we learned that this collection can no longer be found.

Among many challenges that we have identified, focused crawling is one of them. The complexity lies in the crawler's correctness and performance, that is, we have to make sure that only highly relevant webpages should be collected and the crawler should run fast, efficiently, and incrementally because of the potential huge amount of webpage data. We are fortunate that we could reuse Mohamed's crawling engine, and so we were more confident than before that we could handle this task successfully. Another challenge that Dr. Fox pointed out is the issue of redundancy and recency of data. Specifically, it is possible that multiple URLs that are already in HBase might correspond to the same webpage. Even in the case where a URL is repeated, we have to make sure that each is fetched only once. The reason for this is that the webpages corresponding to the URLs might be recently updated and just need to be fetched. To deal with this problem, Dr. Fox suggested us to use available tools such as Heritrix [12] and Archive-It. Since then we have been researching and doing some experiments with the tools to apply them in our situation. In addition, we are consulting the relevant former groups to see if we can reuse any existing code implementing the crawling component.

In the next section, we will discuss related work that is closely relevant to our team's by conducting a literature review.

2. Literature Review

We have found Mohamed Farag's dissertation [1] and the report of the Collection Management team of the Spring 2016 semester [2] the most useful in initially understanding our problem.

Mohamed Farag's dissertation details the focused crawler we will be integrating with our web page collection management system. The event model we will have to construct for each event consists of a vector containing key terms, locations, and a date. The process of seed URL selection is also outlined as grouping URLs by domain/source, sorting the domains by frequency of URLs, and selecting the top k sources. URLs will be sourced from tweets classified by the CLA team as relevant to a particular real world event, e.g., Hurricane Isaac. We will have to perform this process for each event, and incrementally add seed URLs as URLs are aggregated into HBase [4]. Other URLs and webpage sources that we have investigated include the set of 65 webpage collections [5] hosted by Archive-It [6].

The report of the Spring 2016 semester Collection Management group details the current column families in HBase related to the web page collection. The full list of column headers can be found in Table 1. Additionally, the user and developer manual sections will be useful in evaluating their code that is responsible for URL expansion, duplicate removal, web page fetching, and information extraction.

We have also found the course textbook [3] chapters 3, 10, 19, 20, and 21 relevant to our tasks of focused crawling and noise reduction in the form of information extraction from web pages.

3. Requirements and Tasks

The following is a list of requirements and tasks that our final Webpage Collection Management system must meet:

3.1 HTML Fetching

• Filter duplicate URLs across different collections produced by the Tweet Management team as well as our own focused crawler runs. URLs will be read from the clean-tweet and webpage column families in the class HBase table.

• Fetch the HTML content of URLs. This process should run incrementally and on the cluster due to the time cost.
• Store the fetched HTML content in the webpage column family. This process must run on the cluster and in a distributed manner due to memory limits on the cluster driver and the size of the fetched HTML content.
• Add timestamps corresponding to the time the HTML content for URLs that are fetched to accommodate re-fetching after an amount of time. This is done to preserve the freshness of the webpage collections.

3.2 WARC Files

• Create a workflow that generates WARC files for webpages sourced from the focused crawler and any URLs extracted from tweets.
• Save and document the generated WARC files as well as any other WARC file collections newly built at Virginia Tech for eventual upload to the Internet Archive.
• Create a workflow for downloading WARC files hosted on archive-it.org, extracting the information outlined in the HBase schema, and storing the results in HBase for future classification.

3.3 HTML Parsing

• Evaluate the solution provided by the Spring 2016 Collection Management team that is responsible for HTML parsing. This entails getting it to run, timing runs, and determining if it can run incrementally.
• Augment or replace the existing solution to parse additional information outlined in the HBase schema.
• For the webpages associated with the valid and expanded URLs, store the raw HTML, remove the advertising content, banners and other such content from the HTML page and only keep the clean text to be processed, and other relevant web page information. Store the cleaned webpage in HBase.

3.4 Focused Crawler

• Install the focused crawler developed by Mohamed on the EFC2 machine.
• Perform focused crawler runs using our own topic models and seed URLs.
• Store the crawled URLs in HBase for HTML fetching.
• Evaluate the focused crawler for precision, recall, and F1 score. These metrics depend on manually identifying a target number of pages sought by the focused crawler in advance. Accordingly, since we may not know how many pages might relate to an event of interest, we also will use the harvest ratio measure.
• Modify Mohamed's focused crawler to generate WARC files and save the HTML content of webpages in addition to the list of crawled URLs.
• Create a workflow for performing focused crawler runs asynchronously, and incrementally; e.g, focused crawl climate change and shootings at the same time using different crawlers. Pause each crawler as necessary due to resources or waiting for the pipeline of loading processed webpages into HBase. Restart focused crawlers whenever enough new data has arrived.

4. System Design

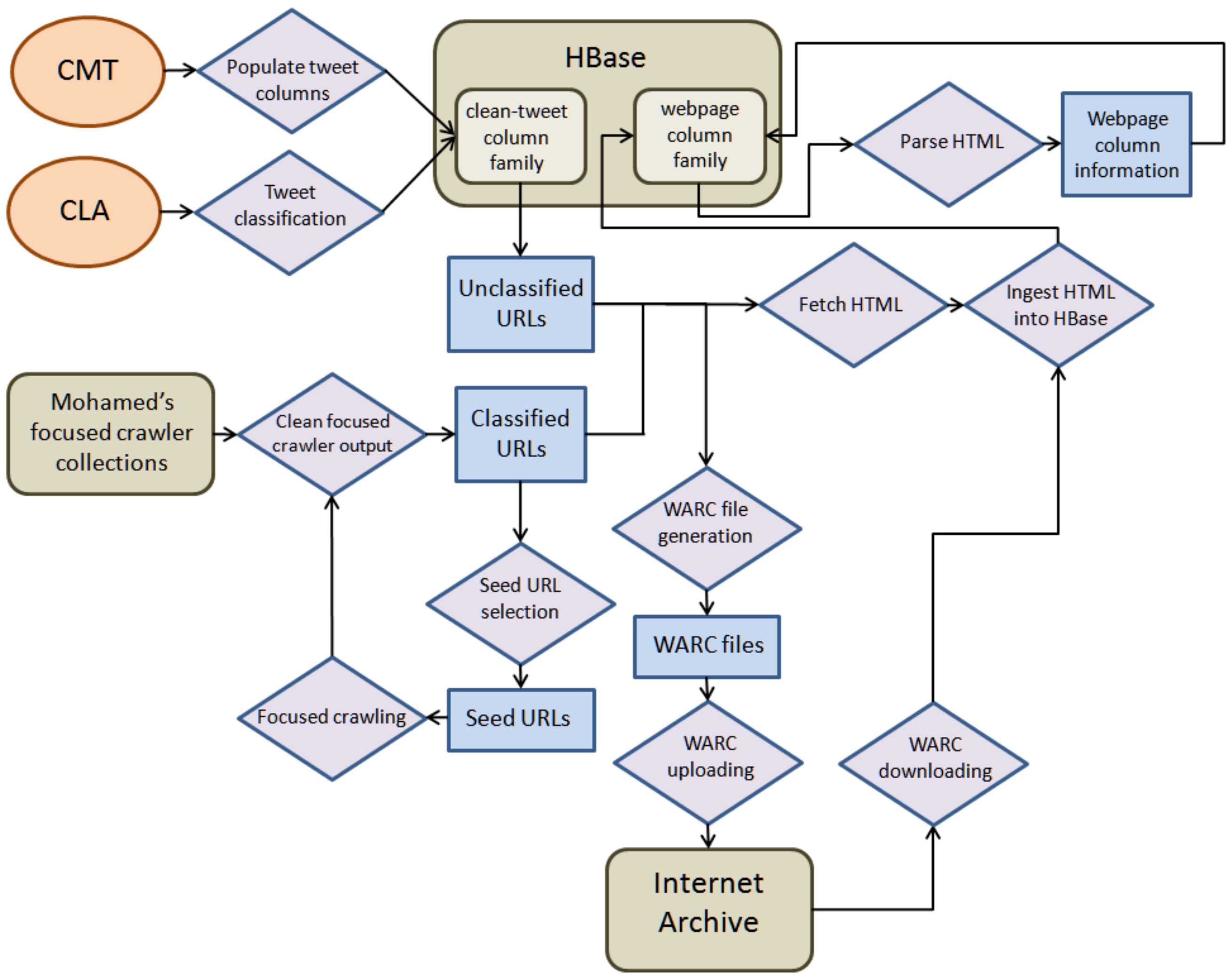


Figure 1: System Pipeline

Figure 1 is a visual representation of our system flow. There are three sources of data: collections of URLs produced by Mohamed's focused crawler, WARC files hosted on the Internet Archive, and the class HBase table. The individual components are explained in the following sections.

4.1 Collaborators

CMT: The CMT team is responsible for populating the tweet column families in the class HBase table. Our team will consume the "long-url" column, under the "clean-tweet" column family, which consists of expanded URLs linked by tweets. For each URL in this column, we will generate a WARC record to eventually be incorporated into a WARC file of the corresponding collection, parse the information required by the webpage column family as specified by the webpage column family schema (Table 2), and store the results in HBase.

CLA: The CLA team is responsible for assigning classification labels to tweets and storing them in the "classification-label" column under the "clean-tweet" column family. For each URL, we process in the "long-url" column, if there are any classification labels assigned, we will store a value of "1" in the "classification-tag" column found under the "webpage" column family to indicate the webpage has been classified at some point in the system. This information is required by the teams who consume the "webpage" column family.

CTE/FE/SOLR: These teams will consume the information contained in the "webpage" column family for their respective tasks. The webpage schema (Table 2) will serve as the interface between our teams.

4.2 Data Sources and Outputs

HBase: As explained above in Section 4.1, our team will consume the "long-url" column, under the "clean-tweet" column family. We will also store the raw HTML and processed information specified in the webpage schema (Table 2) in the class HBase table. The HBaseInteraction scripts found on Canvas will be used to store and read values from the HBase table.

Mohamed's focused crawler collections: Three output directories were made available to us corresponding to three focused crawler runs. Each output directory contained the clean text of 500 webpages and their respective URLs. Oddly, some of the clean text records consisted of 404 error messages. We are not sure why these were included in the output.

Internet Archive: There are 66 collections of WARC files containing a total of 323,706 webpages hosted at the following URL: https://archive-it.org/organizations/156

These collections were produced using the Heritrix web crawler and contain lots of noise. We will create a workflow for downloading these collections and storing their contained HTML in HBase. We will mark the corresponding "classification-tag" column as "0" indicating these records require classification.

In addition to using the Internet Archive as a source of webpages, our team is responsible for generating and uploading WARC files for URLs linked by tweets, Mohamed's focused crawl collections, and the results of our own focused crawler runs.

4.3 Processes

Table 1: System Processes

Process	Input	Output	Description	Tools
HTML Fetching	URLs from HBase	HTML content stored in HBase	Fetch the HTML content of webpages belonging to corresponding URLs populated in HBase	Spark Scala
WARC file generation:	URLs from HBase, Mohamed's focused crawler runs, and our own focused crawler runs	WARC files	This process accomplishes two goals: fetching HTML of a URL, and generating the corresponding WARC file.	Python
Uploading WARC files:	WARC files our team has generated	Collections hosted on archive-it.org	Upload generated WARC files to an appropiate collection on archive-it.org corresponding to the source of the WARC file.	TBD
Downloading WARC files from IA:	WARC collections hosted on archive-it.org	WARC files	Collections can be downloaded using wget, collection source must be preserved.	Command documented at https://webarchive.jira.com/wiki/display/ARIH/Quick-Start+for+Partners+with+Access+to+a+Unix+or+Cygwin+Environment
Ingesting WARC files into HBase:	WARC files we have generated and WARC files from archive-it.org	Raw HTML stored in the "html" column in HBase	warcbase can be used to ingest WARC file to a table in HBase, pig script used for transferring HTML to class HBase table	warcbase, HBaseInteraction pig script
Parsing HTML:	Raw HTML stored in the	Populated "webpage"	Extraction involves parsing HTML tree,	Code documented

	"html" column in HBase	column family in HBase	running SNER on text, removing profanity	in Section 7.3
Seed URL selection:	URLs linked by classified tweets	Set of seed URLs to use as input to focused crawler	Sort URLs by domain, pick sites from top k most frequent domains	Pig or Python script
Focused crawling:	Seed URLs	Focused crawled set of URLs	Involves starting/stopping of focused crawl runs	Documented in Section 7.2
Cleaning focused crawler output:	Focused Crawled URLs	Set of cleaned URLs, (no 404 pages)	Clean the resulting URLs of pages that can't be reached	Pig or Python script

Table 1 serves as an overview of the processes our system is responsible for. The processes highlighted in green have existing work to various degrees while processes in white required novel solutions. Our approach and results of each process are explained in sections 4.5 - 4.9.

4.4 Webpage schema

The full class schema can be found at the following URL: https://docs.google.com/document/d/1sXVyObmfCYmu0PAW2hHDtl4Z--RIyXIRu0VJR9tfRVQ/edit

The schema for webpage information is outlined in Table 2 below. Blue HBase columns were parsed by the Collection Management Team of previous semesters, while white HBase columns indicate new information we are parsing this semester.

Table 2: Webpage schema

Column	Description	Example	Stored	Indexed	Facet
webpage-id	Unique identifier for the webpage	39997223	Yes	id	No
url	URL of the corresponding webpage	http://www.bbc.com/news/world-europe-15551998	Yes	url_s	No

collection-id	number of the collection	651	No	N/A	No
collection-name	name of the collection	electricity	Yes	collection_name_s	Yes
html	raw HTML of web page	[raw HTML text]	No	N/A	No
tweet-ids	unique identifiers of the tweets that contains the URL of this web page,	593392960886145024	No	N/A	No
language	webpage's main language	en	Yes	language_s	Yes
title	extract title from the webpage	Student arrested after threatening Virginia Tech Yik Yak post	Yes	title_s	No
author	extract author from the webpage	Tom LoBianco and Pamela Brown, CNN	Yes	author_s	Yes
created-time	extract created-time from the webpage	Mon Apr 13 19:00:21 +0000 2015	Yes	created_time_dt	Yes
clean-text			No	N/A	No
clean-text-profanity	clean text with no profanity	[clean HTML text]	Yes	text_t	No
sub-urls	sub urls in the webpage		No	sub_urls_s	No
domain-name	extract the domain name from the webpage	http://www.fs.fed.us/	No	N/A	No
domain-location	extract the country name from the webpage	us	Yes	location_s	Yes
organization-name	extract the organization name from the webpage with the help of ©	Cable News Network	Yes	organization_s	Yes
fetched-timestamp	fetched time (readable)	Mon Apr 13 19:00:21 +0000 2015	No	fetched_time_dt	No
event	a list of events in the webpage	Hurricane Matthew; Flood	YES	events_s	NO
classification-tag	identify whether the web page has been previously classified or not	0 / 1	No	N/A	No
webpage_importance	The importance value of each webpage	[0 - 1]	No	w_importance_f	No

4.5 HTML Fetching

The HTML fetching component of our system is responsible for taking URLs produced by the focused crawler or URLs stored in HBase by the CMT team and fetching the HTML content of the corresponding webpage. There is a significant time cost involved in retrieving the HTML of a webpage due to several factors including DNS lookup, and geographical distance to the webpage host. This time cost motivated the use of the DLRL cluster to run HTML fetching in parallel. We could not find any existing tools for HTML fetching that ran in a distributed manner such as an Apache Spark application or MapReduce job.

In developing our own distributed HTML fetching application, we turned to Apache Spark because it provides a function for retrieving the HTML of a URL, something that would require additional libraries if we were to write a MapReduce job. The final developed component can be found in the "htmlFetching" folder of our included code, and its usage is explained in the User Manual section. The component we developed takes a line delimited list of URLs in a text file as input and reads them as a Spark Resilient Distributed Dataset. The HTML content is then fetched in parallel. While ideally the Spark application would read URLs directly from our class HBase table, bugs in the Spark methods to handle HBase reading as well as time constraints prevented us from achieving this. The same goes for the output of the developed Spark application, which in its current form is a string delimited text file of HTML content. An example of the output is given in the User Manual.

Table 3 shows the runtime of the Spark application running on a single driver. We did not run experiments on distributed runs of the Spark application due to time constraints; however, these runtimes at least give some idea of the time required to fetch HTML on a single node.

In terms of future work, future teams should look into different methods of reading and writing the URLs and HTML content directly into HBase. The Spark application should also use an additional column in the "clean-tweet" column family consisting of a binary value corresponding to whether or not the webpage of the contained URL has been fetched when compiling the list of URLs to fetch in an incremental manner.

Table 3: HTML Fetching Timings

Number of URLs	Spark job runtime (seconds)
64	23.031
128	10.752
256	16.876
512	38.756

4.7 WARC File Generation

Web Archive (WARC) files are used by the Internet Archive to store information harvested during web crawls. They are also commonly used as a format for hosting collections of webpages. A WARC file contains set of archived WARC records where each record contains the information for serving an individual resource of a website such as the index.html webpage, or any individual embedded image or audio file of a website. The entire content of a website or collection of webpages can be stored as a collection of WARC records archived in a single WARC file [19].

There is a significant time cost involved in generating the WARC file required to mirror a single website which varies in how content-rich a particular website is. The time to generate a WARC file for a single webpsite is often on the scale of minutes.

There are many existing tools for generating WARC files; however, many of these tools are implemented as web crawlers. While these tools can usually be set not to follow links, the web crawling functionality is unneeded in the context of our system. Additionally, none of the existing tools we found ran in parallel as a Spark application or otherwise. This motivated us to write our own.

The developed application can be found in the "warcGeneration" folder of our included project code and is explained in the User Manual section. We chose to implement the application as a Python script because we could not find any already-installed libraries for either Spark or MapReduce for generating WARC files; thus, the application is not distributed. Instead, the Python script calls the shell command wget, a GNU package for retrieving web resources which includes options for producing WARC files. Details of the script can be found in the Developer Manual section.

4.8 WARC File Ingestion

Unfortunately, due to time constraints we did not focus on WARC file ingestion. However, there is an existing tool called warcbase that takes a collection of WARC files as input and stores them in HBase. Andrej Galad, a student in a previous semester of the class, modified this tool to run on the DLRL cluster [18]. This tool stores the WARC information as a byte array in an HBase table with a specific schema. Details on extending this work for tighter integration with the class system as a whole can be found in the Developer Manual section.

4.9 HTML Parsing and Interaction with HBase

Because of the diversity of our sources of input, one of our workflows starts with URLs from the CMT group that are stored in the HBase. These URLs need to be fetched to obtain the corresponding HTML files. Then, the HTML files are processed and stored back to the HBase table. This workflow requires interacting with HBase, the Internet, and the extracted HTML files.
The first step is loading HTML data from the webpages column family from HBase which we accomplished by using an Apache Pig script. Details about this work are discussed in the User and Developer Manual.

Next, our workflow involves parsing and noise reduction. We have generated a Python script to accomplish this and generate the data we need in .avro file format, an improvement from last semester’s group.
Finally, we load the results back to HBase to complete the information storage part. Here, we employ the Apache Pig script again which is discussed in the User Manual and Developer Manual.

5. Project Plan and Schedule

Table 4: Webpage schema

Date	Task Description
09/06	Installed Virtual Box: Downloaded and deployed Virtual Machine on our own laptop or computer.
09/12	Connected the cluster successfully and read the instruction of the Tutorials files
09/20	Came up with the overall architecture and submitted Interim Report 1
09/26	Implementing the short URLs expanding modules on Hadoop and evaluating the efficiency of different tools packages.
10/03	Fetch a couple of URLs in HBase and loading the raw webpages back to HBase
10/10	Choose the most appropriate tools (for example, BeautifulSoup) and utilize them to clean the loaded webpages file (WARC file or directly textual file from Mohamed’s code) and load them to HBase column.
10/11	Submit Interim Report 2
10/18	Implement the event focused crawling code from Mohamed and try to improve it for our project tasks
10/24	Solve the timestamps adding issue
10/31	Collect the feedback from other groups and, time permitting, try to load webpages from previous and other sources
11/01	Submit Interim Report 3
11/07	Get all additional implementation done and start running our code/tool on the DLRL cluster

11/14	Keep running our code to collect webpages to HBase
12/01 & 12/06	Final project presentation
12/08	Submit Final project report & source code

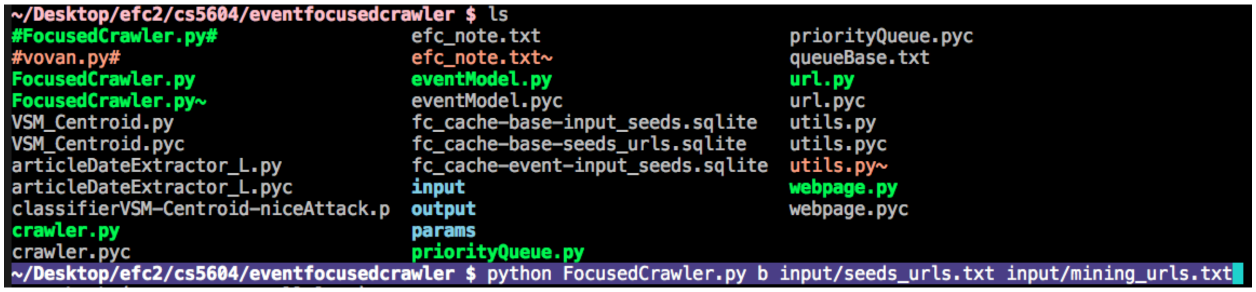
6. Implementation and Experiments

6.1 Experiments with Focused Crawler

In this section, we will describe our experiments and their preliminary results involving the focused crawler. For the sake of simplicity, we set the crawler to run on a small set of URLs with a termination condition corresponding to the number of pages to crawl. This, however, does not prevent us from scaling it up to run on a large amount of data incrementally and continuously on the DLRL cluster later.

6.1.1 Settings and Input Data

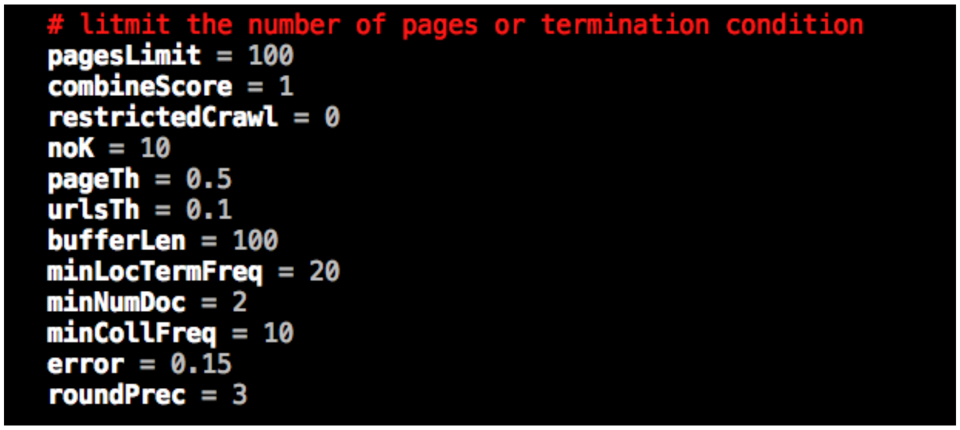
In this experiment, we were interested in the event "South China Sea Disputes", and we wanted the crawler to search for the webpages that are relevant to the event. In order for the crawler to achieve this goal, first we needed to prepare a training data set of seed urls of five webpages that are as relevant to the event as possible. We did this by manually collecting URLs, and saving them to an input text file called *seed_urls.txt*. Once trained, the crawler was ready to search for and archive relevant webpages, given a small set of other seed URLs in a file called *mining_urls.txt.* Both operations could be run with the following command (the -*b* parameter means the algorithm for crawling is a baseline method):



```
~/Desktop/efc2/cs5604/eventfocusedcrawler $ ls
#FocusedCrawler.py#                 efc_note.txt                      priorityQueue.pyc
#vovan.py#                          efc_note.txt~                     queueBase.txt
FocusedCrawler.py                   eventModel.py                     url.py
FocusedCrawler.py~                  eventModel.pyc                    url.pyc
VSM_Centroid.py                     fc_cache-base-input_seeds.sqlite  utils.py
VSM_Centroid.pyc                    fc_cache-base-seeds_urls.sqlite   utils.pyc
articleDateExtractor_L.py           fc_cache-event-input_seeds.sqlite utils.py~
articleDateExtractor_L.pyc          input                             webpage.py
classifierVSM-Centroid-niceAttack.p output                            webpage.pyc
crawler.py                          params
crawler.pyc                         priorityQueue.py
~/Desktop/efc2/cs5604/eventfocusedcrawler $ python FocusedCrawler.py b input/seeds_urls.txt input/mining_urls.txt
```

Figure 2: Event Focused Crawler Command Line

The crawler was configured as follows:

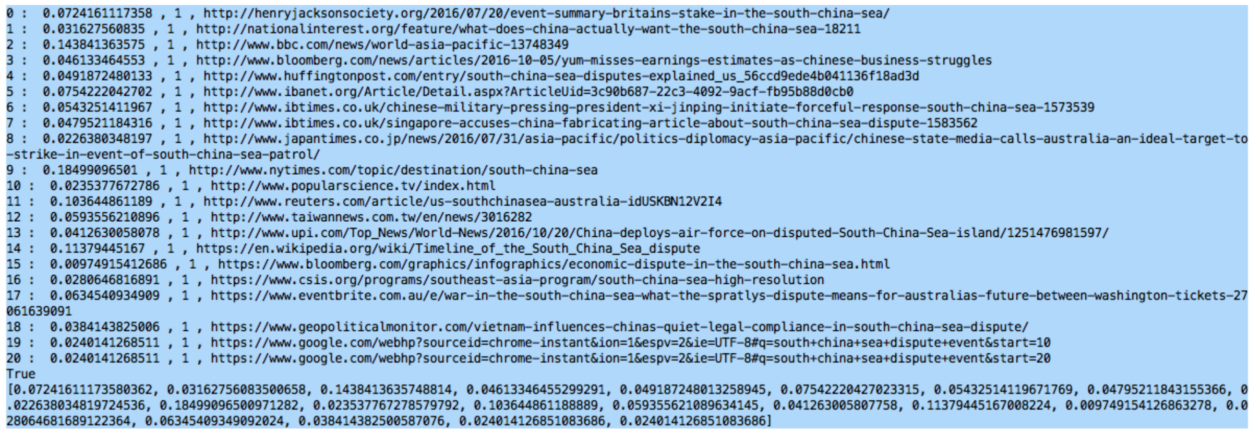


```
# litmit the number of pages or termination condition
pagesLimit = 100
combineScore = 1
restrictedCrawl = 0
noK = 10
pageTh = 0.5
urlsTh = 0.1
bufferLen = 100
minLocTermFreq = 20
minNumDoc = 2
minCollFreq = 10
error = 0.15
roundPrec = 3
```

Figure 3: Configuration of Event Focused Crawler

6.1.2 Results

The crawler would output a ranked list of relevant URLs (URL - and its corresponding relevancy score), as shown in the below figure.

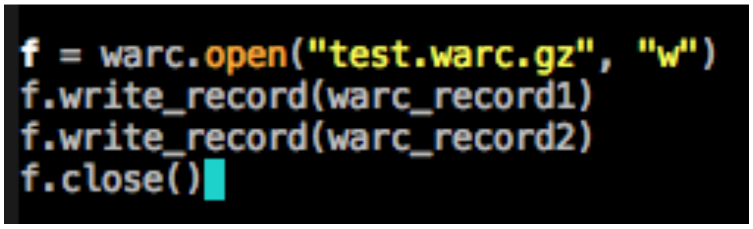


```
0 :  0.0724161117358 , 1 , http://henryjacksonsociety.org/2016/07/20/event-summary-britains-stake-in-the-south-china-sea/
1 :  0.031627560835 , 1 , http://nationalinterest.org/feature/what-does-china-actually-want-the-south-china-sea-18211
2 :  0.143841363575 , 1 , http://www.bbc.com/news/world-asia-pacific-13748349
3 :  0.046133464553 , 1 , http://www.bloomberg.com/news/articles/2016-10-05/yum-misses-earnings-estimates-as-chinese-business-struggles
4 :  0.0491872480133 , 1 , http://www.huffingtonpost.com/entry/south-china-sea-disputes-explained_us_56ccd9ede4b041136f18ad3d
5 :  0.0754222042702 , 1 , http://www.ibanet.org/Article/Detail.aspx?ArticleUid=3c90b687-22c3-4092-9acf-fb95b88d0cb0
6 :  0.0543251411967 , 1 , http://www.ibtimes.co.uk/chinese-military-pressing-president-xi-jinping-initiate-forceful-response-south-china-sea-1573539
7 :  0.0479521184316 , 1 , http://www.ibtimes.co.uk/singapore-accuses-china-fabricating-article-about-south-china-sea-dispute-1583562
8 :  0.0226380348197 , 1 , http://www.japantimes.co.jp/news/2016/07/31/asia-pacific/politics-diplomacy-asia-pacific/chinese-state-media-calls-australia-an-ideal-target-to
-strike-in-event-of-south-china-sea-patrol/
9 :  0.18499096501 , 1 , http://www.nytimes.com/topic/destination/south-china-sea
10 :  0.0235377672786 , 1 , http://www.popularscience.tv/index.html
11 :  0.103644861189 , 1 , http://www.reuters.com/article/us-southchinasea-australia-idUSKBN12V2I4
12 :  0.0593556210896 , 1 , http://www.taiwannews.com.tw/en/news/3016282
13 :  0.0412630058078 , 1 , http://www.upi.com/Top_News/World-News/2016/10/20/China-deploys-air-force-on-disputed-South-China-Sea-island/1251476981597/
14 :  0.11379445167 , 1 , https://en.wikipedia.org/wiki/Timeline_of_the_South_China_Sea_dispute
15 :  0.00974915412686 , 1 , https://www.bloomberg.com/graphics/infographics/economic-dispute-in-the-south-china-sea.html
16 :  0.0280646816891 , 1 , https://www.csis.org/programs/southeast-asia-program/south-china-sea-high-resolution
17 :  0.0634540934909 , 1 , https://www.eventbrite.com.au/e/war-in-the-south-china-sea-what-the-spratlys-dispute-means-for-australias-future-between-washington-tickets-27
061639091
18 :  0.0384143825006 , 1 , https://www.geopoliticalmonitor.com/vietnam-influences-chinas-quiet-legal-compliance-in-south-china-sea-dispute/
19 :  0.0240141268511 , 1 , https://www.google.com/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=south+china+sea+dispute+event&start=10
20 :  0.0240141268511 , 1 , https://www.google.com/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=south+china+sea+dispute+event&start=20
True
[0.07241611173580362, 0.03162756083500658, 0.1438413635748814, 0.04613346455299291, 0.04918724801325894 5, 0.07542220427023315, 0.05432514119671769, 0.04795211843155366, 0
.022638034819724536, 0.18499096500971282, 0.023537767278579792, 0.103644861188889, 0.059355621089634145, 0.041263005807758, 0.11379445167008224, 0.009749154126863278, 0.0
28064681689122364, 0.06345409349092024, 0.038414382500587076, 0.024014126851083686, 0.024014126851083686]
```

Figure 4: Outputs of Event Focused Crawler

6.1.3 On-going Work

There are two main extensions needed to improve the crawler. Currently, the crawler saves the relevant output webpages in a text file. Our goal from the beginning is to archive the collected webpages in the form of WARC files as well as store their HTML content in HBase. Therefore, we need to implement this feature for the crawler. This task can be done relatively easily in Python with, for example, the WARC library and warcbase; or in Java with Heritrix. For example, to write to a warc file, in Python we can do:



```
f = warc.open("test.warc.gz", "w")
f.write_record(warc_record1)
f.write_record(warc_record2)
f.close()
```

Figure 5: WARC Operation in Python

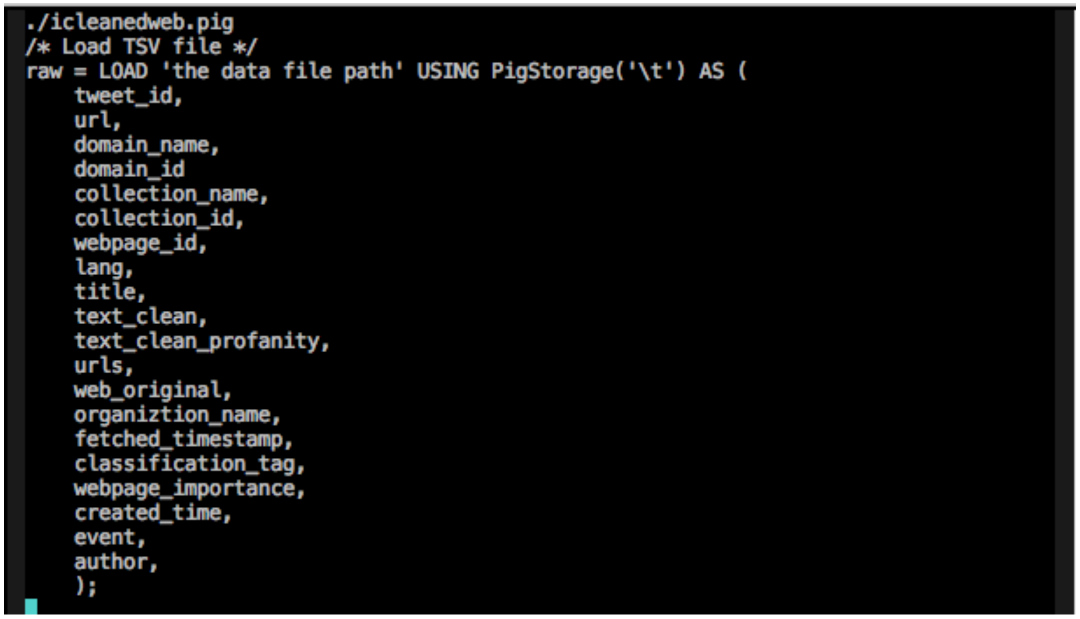
The second extension is to extract information fields (e.g., author, date, location, organization) from a collected web page. Again, this feature can be easily implemented using the Stanford NER framework, and/or Beautifulsoup.

Finally, the last work with this crawler is to run it on the DLRL cluster to collect and save webpages to HBase.

6.2 Interacting with HBase: Pig Script

Apache Pig is a high-level platform for creating programs that run on Apache Hadoop. The language for this platform is called Pig Latin. Pig can execute its Hadoop jobs in MapReduce, Apache Tez, or Apache Spark. Pig Latin abstracts the programming from the Java MapReduce idiom into a notation which makes MapReduce programming high level, similar to that of SQL for RDBMSs. Pig Latin can be extended using User Defined Functions (UDFs) which the user can write in Java, Python, JavaScript, Ruby or Groovy and then call directly from the language. Due to those reasons, we chose Pig as the method to interact with HBase in our project. Our Pig script for loading extracted data (in text form) looks like this:

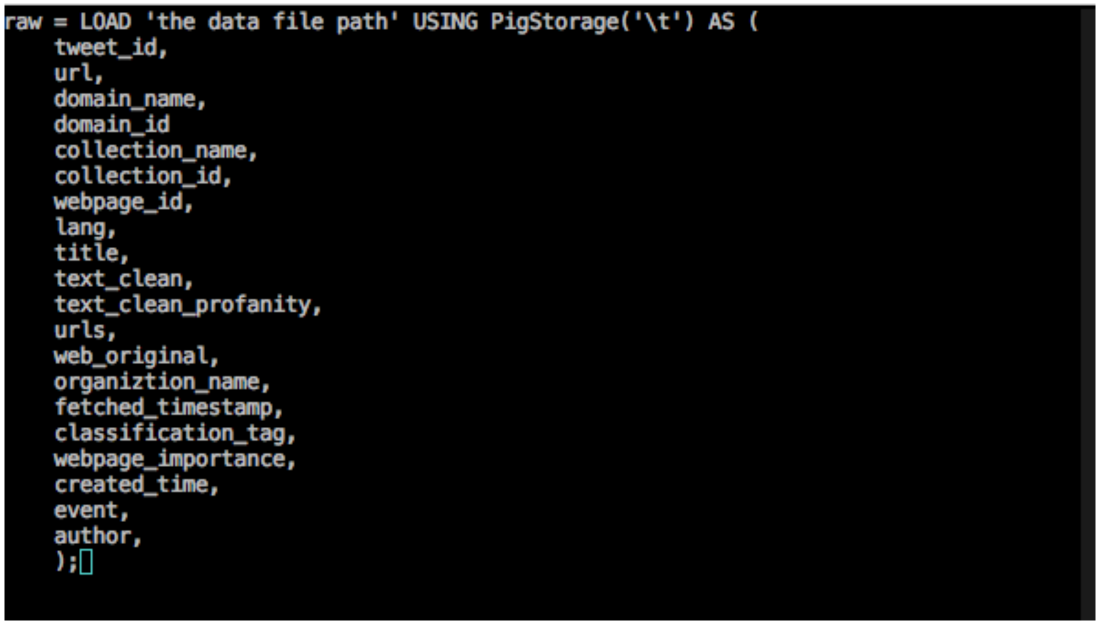
./icleanedweb.pig



```
./icleanedweb.pig
/* Load TSV file */
raw = LOAD 'the data file path' USING PigStorage('\t') AS (
    tweet_id,
    url,
    domain_name,
    domain_id
    collection_name,
    collection_id,
    webpage_id,
    lang,
    title,
    text_clean,
    text_clean_profanity,
    urls,
    web_original,
    organiztion_name,
    fetched_timestamp,
    classification_tag,
    webpage_importance,
    created_time,
    event,
    author,
    );
```

Figure 6: Load Data in Pig

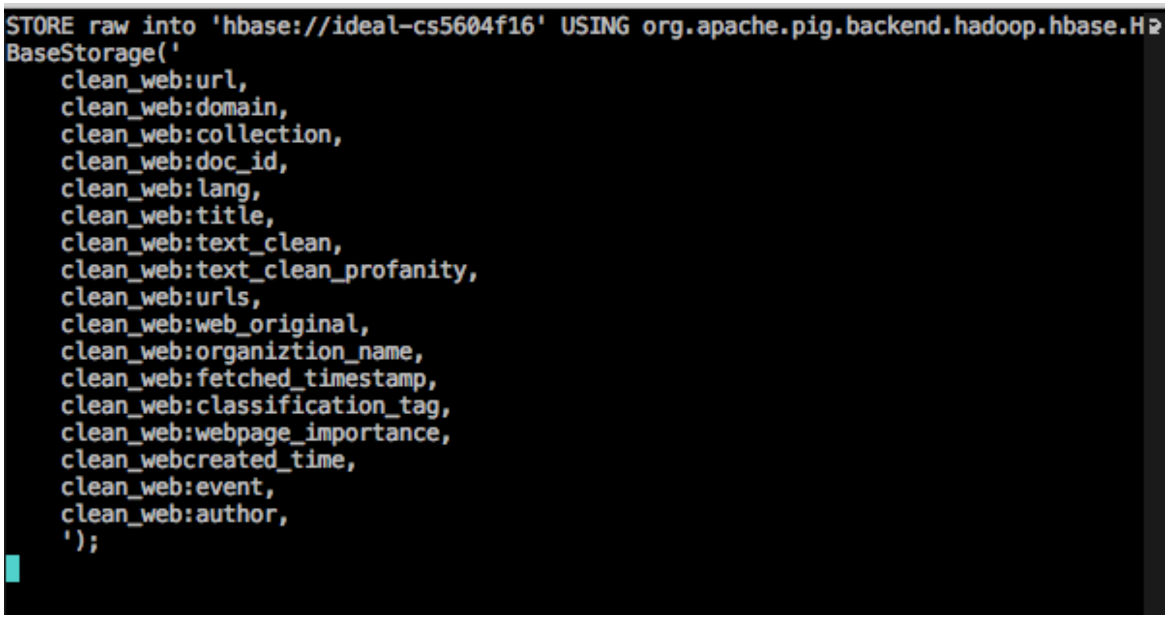
/* Load TSV file */



```
raw = LOAD 'the data file path' USING PigStorage('\t') AS (
    tweet_id,
    url,
    domain_name,
    domain_id
    collection_name,
    collection_id,
    webpage_id,
    lang,
    title,
    text_clean,
    text_clean_profanity,
    urls,
    web_original,
    organiztion_name,
    fetched_timestamp,
    classification_tag,
    webpage_importance,
    created_time,
    event,
    author,
    );
```

Figure 7: Load Data in TSV File

/* Store data into HBase */

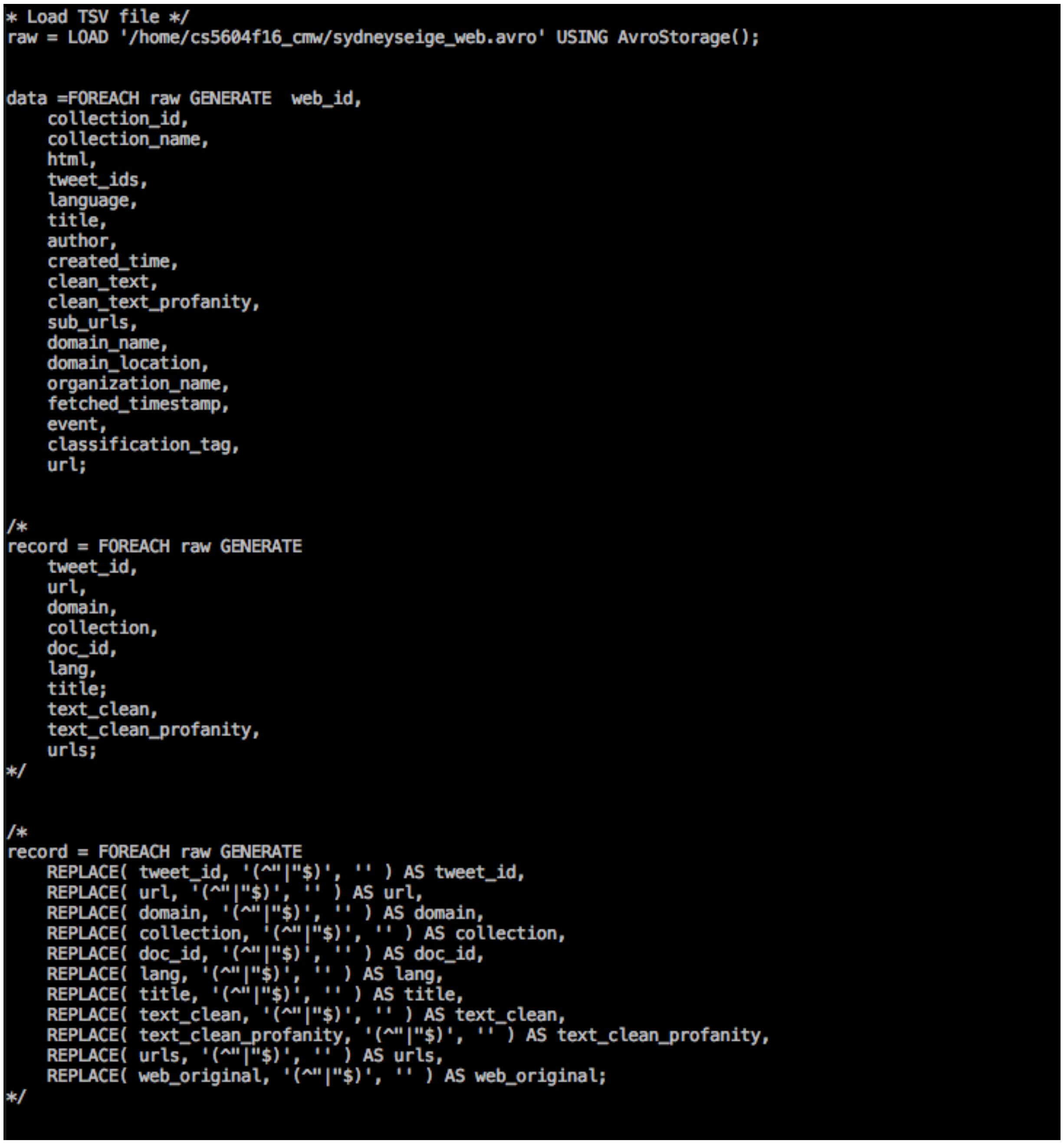


```
STORE raw into 'hbase://ideal-cs5604f16' USING org.apache.pig.backend.hadoop.hbase.H
BaseStorage('
    clean_web:url,
    clean_web:domain,
    clean_web:collection,
    clean_web:doc_id,
    clean_web:lang,
    clean_web:title,
    clean_web:text_clean,
    clean_web:text_clean_profanity,
    clean_web:urls,
    clean_web:web_original,
    clean_web:organiztion_name,
    clean_web:fetched_timestamp,
    clean_web:classification_tag,
    clean_web:webpage_importance,
    clean_webcreated_time,
    clean_web:event,
    clean_web:author,
    ');
```

Figure 8: Store Data into HBase with Pig

The only part that needs to be filled by the user is the data file path, which we have underlined. The bold text is the path for the HBase table we want to load the data into.
The above Pig script is used for loading a pure text data file into HBase, which works better for explaining how a Pig script works. However, it ended up being ineffective and unstable to use text data files because we used '\t' and '\n' to separate different raw records, which may be also included in the raw content themselves.
Therefore we needed to go one step further and employ the .avro file format. The corresponding Pig script uses the .avro file as input to load the data into HBase is shown below:

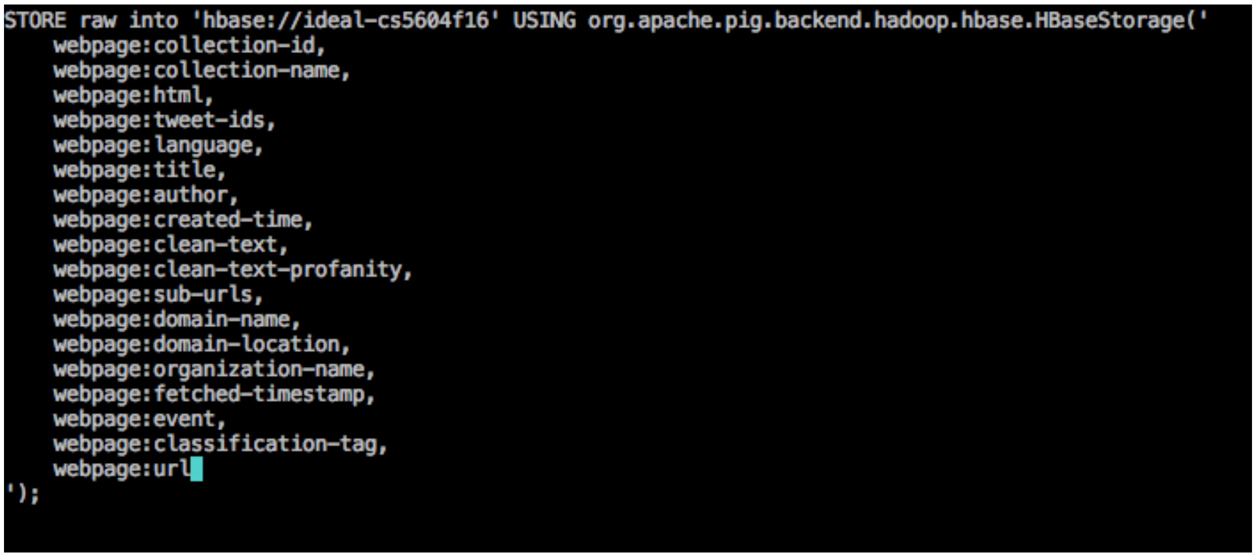
./avroload.pig



```
* Load TSV file */
raw = LOAD '/home/cs5604f16_cmw/sydneyseige_web.avro' USING AvroStorage();


data =FOREACH raw GENERATE  web_id,
    collection_id,
    collection_name,
    html,
    tweet_ids,
    language,
    title,
    author,
    created_time,
    clean_text,
    clean_text_profanity,
    sub_urls,
    domain_name,
    domain_location,
    organization_name,
    fetched_timestamp,
    event,
    classification_tag,
    url;


/*
record = FOREACH raw GENERATE
    tweet_id,
    url,
    domain,
    collection,
    doc_id,
    lang,
    title;
    text_clean,
    text_clean_profanity,
    urls;
*/


/*
record = FOREACH raw GENERATE
    REPLACE( tweet_id, '(^"|"$)', '' ) AS tweet_id,
    REPLACE( url, '(^"|"$)', '' ) AS url,
    REPLACE( domain, '(^"|"$)', '' ) AS domain,
    REPLACE( collection, '(^"|"$)', '' ) AS collection,
    REPLACE( doc_id, '(^"|"$)', '' ) AS doc_id,
    REPLACE( lang, '(^"|"$)', '' ) AS lang,
    REPLACE( title, '(^"|"$)', '' ) AS title,
    REPLACE( text_clean, '(^"|"$)', '' ) AS text_clean,
    REPLACE( text_clean_profanity, '(^"|"$)', '' ) AS text_clean_profanity,
    REPLACE( urls, '(^"|"$)', '' ) AS urls,
    REPLACE( web_original, '(^"|"$)', '' ) AS web_original;
*/
```

Figure 9: Load Data from A TSV File

/* Store data into HBase */



```
STORE raw into 'hbase://ideal-cs5604f16' USING org.apache.pig.backend.hadoop.hbase.HBaseStorage('
    webpage:collection-id,
    webpage:collection-name,
    webpage:html,
    webpage:tweet-ids,
    webpage:language,
    webpage:title,
    webpage:author,
    webpage:created-time,
    webpage:clean-text,
    webpage:clean-text-profanity,
    webpage:sub-urls,
    webpage:domain-name,
    webpage:domain-location,
    webpage:organization-name,
    webpage:fetched-timestamp,
    webpage:event,
    webpage:classification-tag,
    webpage:url
');
```

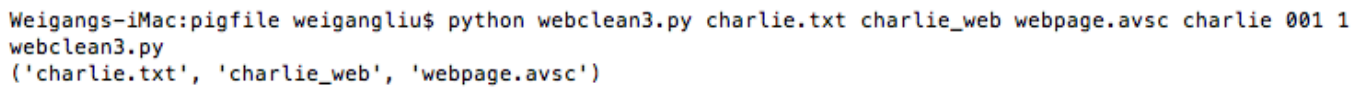
Figure 10: Store Data into HBase

Here, the data structure in the Pig script we define should be consistent with the .avro data file structure we specified in the schema. If one wants to modify the data structure, he or she must load into the HBase table, both the .avro schema (thus the Python extractor script) and the Pig load data script should be changed to be consistent with each other.

7. User Manual

7.1 Webpage Parsing and Clean

To use the webpage clean code mentioned in the Developer Manual, for example we can just use the command:



```
Weigangs-iMac:pigfile weigangliu$ python webclean3.py charlie.txt charlie_web webpage.avsc charlie 001 1
webclean3.py
('charlie.txt', 'charlie_web', 'webpage.avsc')
```

Figure 11: Webclean Script Demo

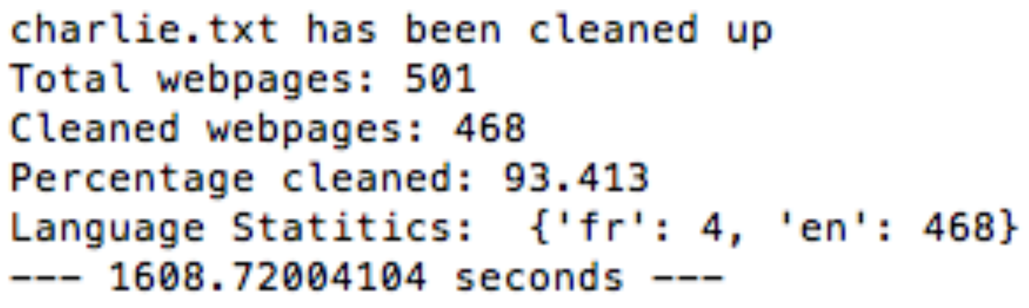
This is the example demo for using the webpage clean Python script: webclean3.py. Here, the input file "charlie.txt" contains the URL list for the event Charlie Hebdo Shooting collection, which comes from the previous research work done by Mohamed Magdy Gharib Farag, the file "charlie_web" is the output text file that contains all the content needed in the webpage schema table, which can then load the results into the HBase table directly using the Pig script in the Developer Manual.

Because we were faced with multiple types of input from different sources such as URLs, WARC files, text files or HTML files, we have different copies of Python scripts for different inputs to avoid additional, manual, adjustment.

Since at this moment the DLRL cluster does not have included the Python libraries for parsing and noise reduction (Such as Beautiful Soup or readability), this procedure is still acceptable. While further improvement might make it possible to run those Python scripts on cluster automatically, we can call all our separate webpage-clean Python scripts from another script to make the decision of which script to employ according to the characteristics of the input data. Due to time constraints, we haven't complete this job.

Because the number of URLs the CMT group will provide, and our focused crawler will obtain from the Internet is unpredictable at this moment, it seems hard to provide an exact estimation about the size of the data set or the number of the documents we will have. If we purely predict from the number of the events on our list, we may have about 26 large collections of documents to be processed after we generate the code successfully.

Here is the simple effectiveness statistic estimation results for previous webpage clean command:

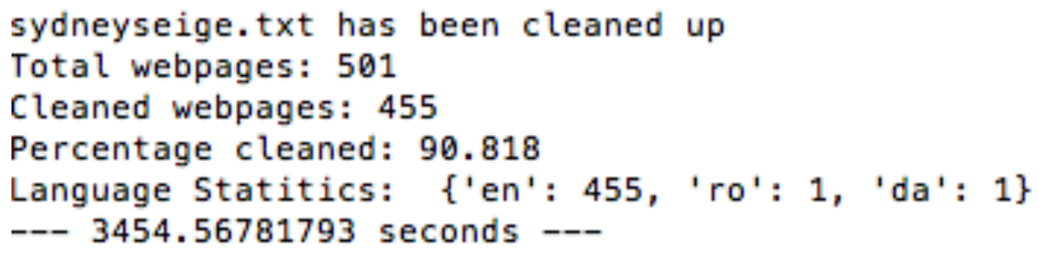


```
charlie.txt has been cleaned up
Total webpages: 501
Cleaned webpages: 468
Percentage cleaned: 93.413
Language Statitics:  {'fr': 4, 'en': 468}
--- 1608.72004104 seconds ---
```

Figure 12: Charlie Hebdo Shooting Collection Clean Statistics

Here we see clearly that our script has a total of 501 URLs, while 468 of them are successfully fetched and cleaned. This experiment is done using an I Mac machine, therefore a large percent of the unfetched URLs are due to the different SSL versions compatibility problem of IOS system. This should not be a problem if we use a Linux machine to run our scripts or ultimately when the scripts can be successfully runned in our cluster.

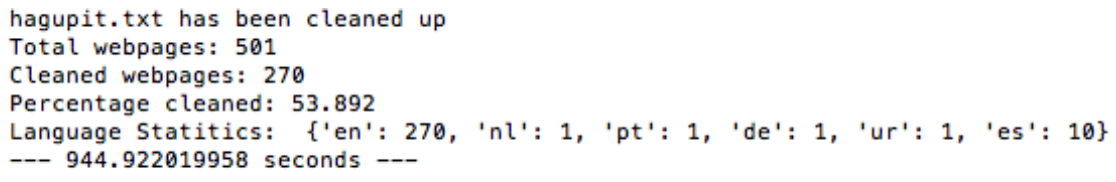
We also have the statistics results for other two test collections here:



```
sydneyseige.txt has been cleaned up
Total webpages: 501
Cleaned webpages: 455
Percentage cleaned: 90.818
Language Statitics:  {'en': 455, 'ro': 1, 'da': 1}
--- 3454.56781793 seconds ---
```

Figure 13: Sydney Hostage Crisis Collection Clean Statistic

This is the webpage fetching and cleaning script results for the 2014 Sydney Hostage Crisis event, which again comes from the focused crawler research collection by Mohamed Magdy Gharib Farag. Here the input is still a URL list.



```
hagupit.txt has been cleaned up
Total webpages: 501
Cleaned webpages: 270
Percentage cleaned: 53.892
Language Statitics:  {'en': 270, 'nl': 1, 'pt': 1, 'de': 1, 'ur': 1, 'es': 10}
--- 944.922019958 seconds ---
```

Figure 14: Hagupit Typhoon Collection Clean Statistic

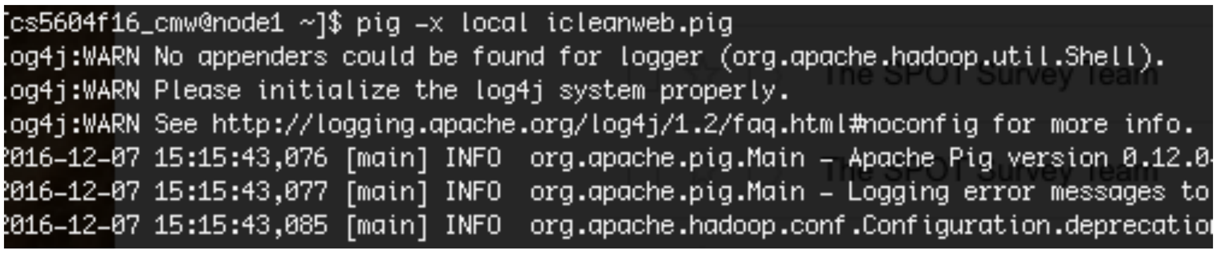
This is the webpage fetching and cleaning script results for the 2014 Typhoon Hagupit which again comes from the focused crawler research collection by Mohamed Magdy Gharib Farag. Here the input is still URL list.

Here we clearly see that for our webpage clean script works fine as we expect it to be.

7.2 Interaction with HBase

7.2.1 Load the webpage data into HBase

To load data into HBase we decided to employ Apache Pig, as will be explained in details in the Developer Manual section. Here we only show the demo of how to use the Pig script.
To fulfill this job we need only one line linux command like:

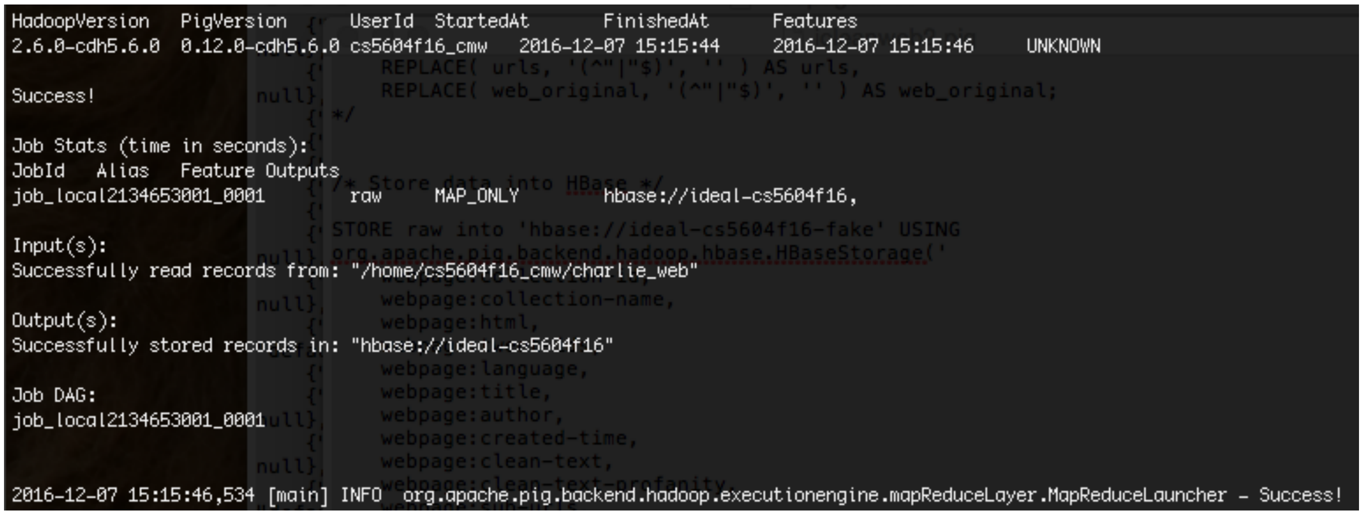


```
[cs5604f16_cmw@node1 ~]$ pig -x local icleanweb.pig
og4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell).
og4j:WARN Please initialize the log4j system properly.
og4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
016-12-07 15:15:43,076 [main] INFO  org.apache.pig.Main - Apache Pig version 0.12.0
016-12-07 15:15:43,077 [main] INFO  org.apache.pig.Main - Logging error messages to
016-12-07 15:15:43,085 [main] INFO  org.apache.hadoop.conf.Configuration.deprecatio
```

Figure 15: Interact with HBase using Pig Script Demo

Here for this demo since we haven’t uploaded our text data file (charlie_web) into HDFS, we only need local mode for the Pig script (the keyword ‘local’ in the command line). If the input data file is large, then we may need the Map Reduce mode and have to load the data files into HDFS first. The second step can be done following the interaction with HBase tutorial in Canvas, however, for the first step we should modify the correct path for the HDFS file and choose the keyword ‘mapreduce’ rather than ‘local’.

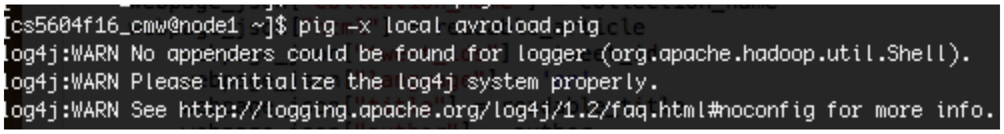
If the loading process is successful, we can view the information in the terminal:



```
HadoopVersion   PigVersion          UserId  StartedAt           FinishedAt          Features
2.6.0-cdh5.6.0  0.12.0-cdh5.6.0 cs5604f16_cmw   2016-12-07 15:15:44     2016-12-07 15:15:46     UNKNOWN

Success!

Job Stats (time in seconds):
JobId   Alias   Feature Outputs
job_local2134653001_0001        raw     MAP_ONLY        hbase://ideal-cs5604f16,

Input(s):
Successfully read records from: "/home/cs5604f16_cmw/charlie_web"

Output(s):
Successfully stored records in: "hbase://ideal-cs5604f16"

Job DAG:
job_local2134653001_0001

2016-12-07 15:15:46,534 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
```

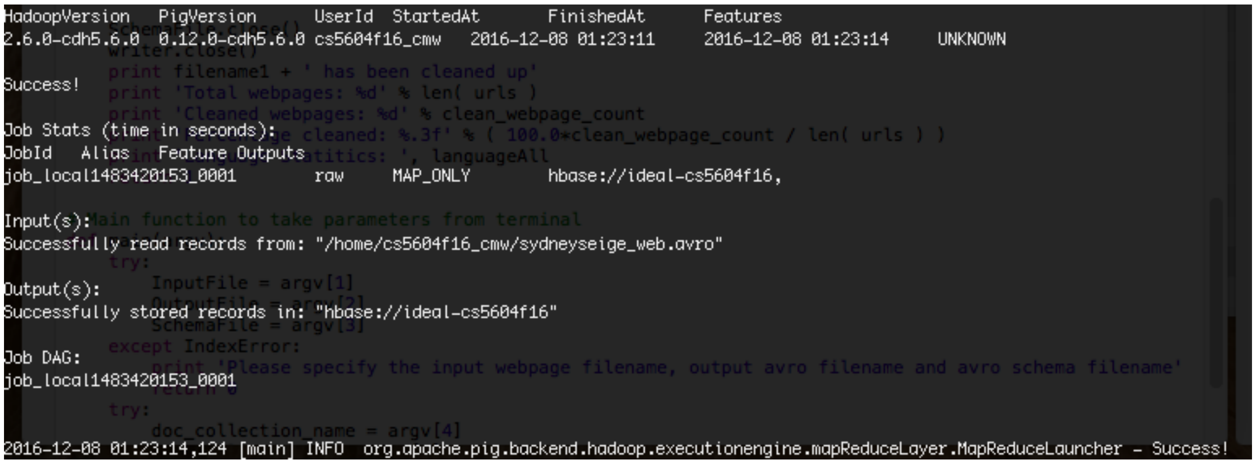
Figure 16: Pig Script Loading Results

Further if the data files are in .avro format, we then need another Pig script to load them into the HBase.
Below we show the Pig script demo to load the .avro file into HBase using local mode, which in some sense looks the same as the text file method. Here the content in the .avro file is not as explicit as the text file, thus the effort needed for users (also developers) to debug the job if errors occur will be harder than the text version.



```
[cs5604f16_cmw@node1 ~]$ pig -x local avroload.pig
log4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
```

Figure 17: Interact with HBase using Pig Script Demo (Avro version)

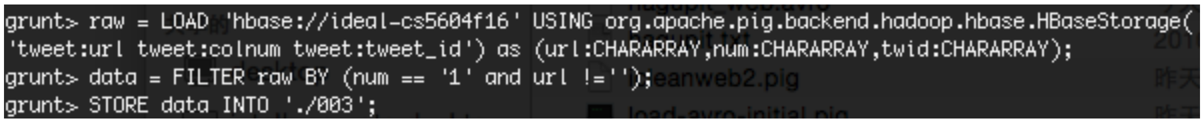


```
HadoopVersion   PigVersion      UserId  StartedAt           FinishedAt          Features
2.6.0-cdh5.6.0  0.12.0-cdh5.6.0 cs5604f16_cmw   2016-12-08 01:23:11     2016-12-08 01:23:14     UNKNOWN

Success!

Job Stats (time in seconds):
JobId   Alias   Feature Outputs
job_local1483420153_0001        raw     MAP_ONLY        hbase://ideal-cs5604f16,

Input(s):
Successfully read records from: "/home/cs5604f16_cmw/sydneyseige_web.avro"

Output(s):
Successfully stored records in: "hbase://ideal-cs5604f16"

Job DAG:
job_local1483420153_0001

2016-12-08 01:23:14,124 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
```

Figure 18: Pig Script Loading Results (Avro version)

7.2.2 Load Input URLs Generated by Tweet Group (CMT) From HBase

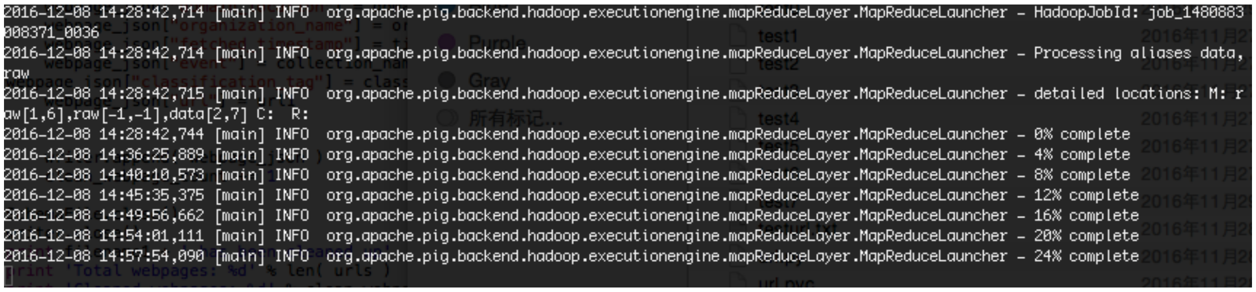
To accomplish this job, we decided to employ the Apache Pig again. While instead of writing a fixed Pig script to fulfill this job, we directly use the command line in the Pig shell to solve this problem, since the data from the CMT group is not static results. It is incrementally updated day by day thus this method is more flexible to deal with this situation.



```
grunt> raw = LOAD 'hbase://ideal-cs5604f16' USING org.apache.pig.backend.hadoop.hbase.HBaseStorage(
'tweet:url tweet:colnum tweet:tweet_id') as (url:CHARARRAY,num:CHARARRAY,twid:CHARARRAY);
grunt> data = FILTER raw BY (num == '1' and url !='');
grunt> STORE data INTO './003';
```

Figure 19: Pig Shell Command Sequence for Loading URLs from HBase

Here we show the demo for how to run this process. In the three lines of commands shown in the above graph, the first line means load the tweets' URL, tweet collection ID and the tweet ID associated with the URL; second one means we filter the results from first line by choosing the collection '1' and its URL searching results are not null; the third line just means we store the filtered results into the current directory/003 in the HDFS.



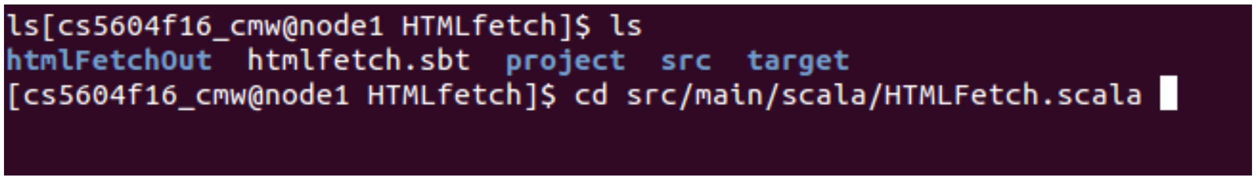
```
2016-12-08 14:28:42,714 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - HadoopJobId: job_1480883
008371_0036
2016-12-08 14:28:42,714 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Processing aliases data,
raw
2016-12-08 14:28:42,715 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - detailed locations: M: r
aw[1,6],raw[-1,-1],data[2,7] C:  R:
2016-12-08 14:28:42,744 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 0% complete
2016-12-08 14:36:25,889 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 4% complete
2016-12-08 14:40:10,573 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 8% complete
2016-12-08 14:45:35,375 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 12% complete
2016-12-08 14:49:56,662 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 16% complete
2016-12-08 14:54:01,111 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 20% complete
2016-12-08 14:57:54,090 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 24% complete
```

Figure 20: Processing Demo for Loading URLs from HBase

Here is an interstage processing demo for the commands above in the Pig shell, we can make a rough prediction that it needs about two hours for those commands to be run by the Pig shell. The generated results should be a series of the text files that contain the URLs, the collection ID of the tweets mentioned those results and those tweets' ID, which are needed information for our webpage cleaning and processing Python script to generate the final data that can be loaded back into HBase.

7.3 HTML Fetching

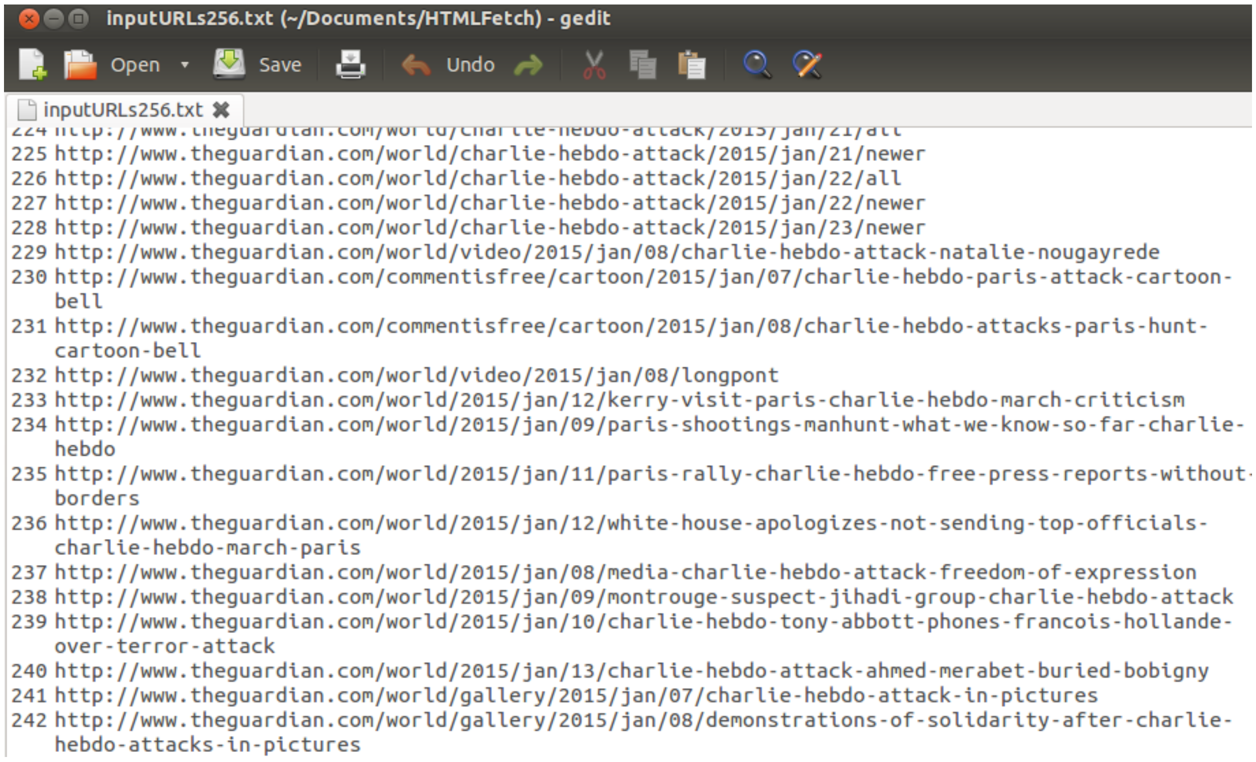
The HTML fetching component is a Spark application and requires a specific directory structure for installation and running shown below in Figure 21.



```
ls[cs5604f16_cmw@node1 HTMLfetch]$ ls
htmlFetchOut  htmlfetch.sbt  project  src  target
[cs5604f16_cmw@node1 HTMLfetch]$ cd src/main/scala/HTMLFetch.scala
```

Figure 21: Spark Directory Structure

The input of the application is a text file of line delimited URLs, an example of which is shown below in Figure 22. This file is stored in HDFS and is specified as an argument to the Spark job command. Several example files are included in the project code.



```
http://www.theguardian.com/world/charlie-hebdo-attack/2015/jan/21/all
http://www.theguardian.com/world/charlie-hebdo-attack/2015/jan/21/newer
http://www.theguardian.com/world/charlie-hebdo-attack/2015/jan/22/all
http://www.theguardian.com/world/charlie-hebdo-attack/2015/jan/22/newer
http://www.theguardian.com/world/charlie-hebdo-attack/2015/jan/23/newer
http://www.theguardian.com/world/video/2015/jan/08/charlie-hebdo-attack-natalie-nougayrede
http://www.theguardian.com/commentisfree/cartoon/2015/jan/07/charlie-hebdo-paris-attack-cartoon-
bell
http://www.theguardian.com/commentisfree/cartoon/2015/jan/08/charlie-hebdo-attacks-paris-hunt-
cartoon-bell
http://www.theguardian.com/world/video/2015/jan/08/longpont
http://www.theguardian.com/world/2015/jan/12/kerry-visit-paris-charlie-hebdo-march-criticism
http://www.theguardian.com/world/2015/jan/09/paris-shootings-manhunt-what-we-know-so-far-charlie-
hebdo
http://www.theguardian.com/world/2015/jan/11/paris-rally-charlie-hebdo-free-press-reports-without-
borders
http://www.theguardian.com/world/2015/jan/12/white-house-apologizes-not-sending-top-officials-
charlie-hebdo-march-paris
http://www.theguardian.com/world/2015/jan/08/media-charlie-hebdo-attack-freedom-of-expression
http://www.theguardian.com/world/2015/jan/09/montrouge-suspect-jihadi-group-charlie-hebdo-attack
http://www.theguardian.com/world/2015/jan/10/charlie-hebdo-tony-abbott-phones-francois-hollande-
over-terror-attack
http://www.theguardian.com/world/2015/jan/13/charlie-hebdo-attack-ahmed-merabet-buried-bobigny
http://www.theguardian.com/world/gallery/2015/jan/07/charlie-hebdo-attack-in-pictures
http://www.theguardian.com/world/gallery/2015/jan/08/demonstrations-of-solidarity-after-charlie-
hebdo-attacks-in-pictures
```

Figure 22: HTML Fetching Input

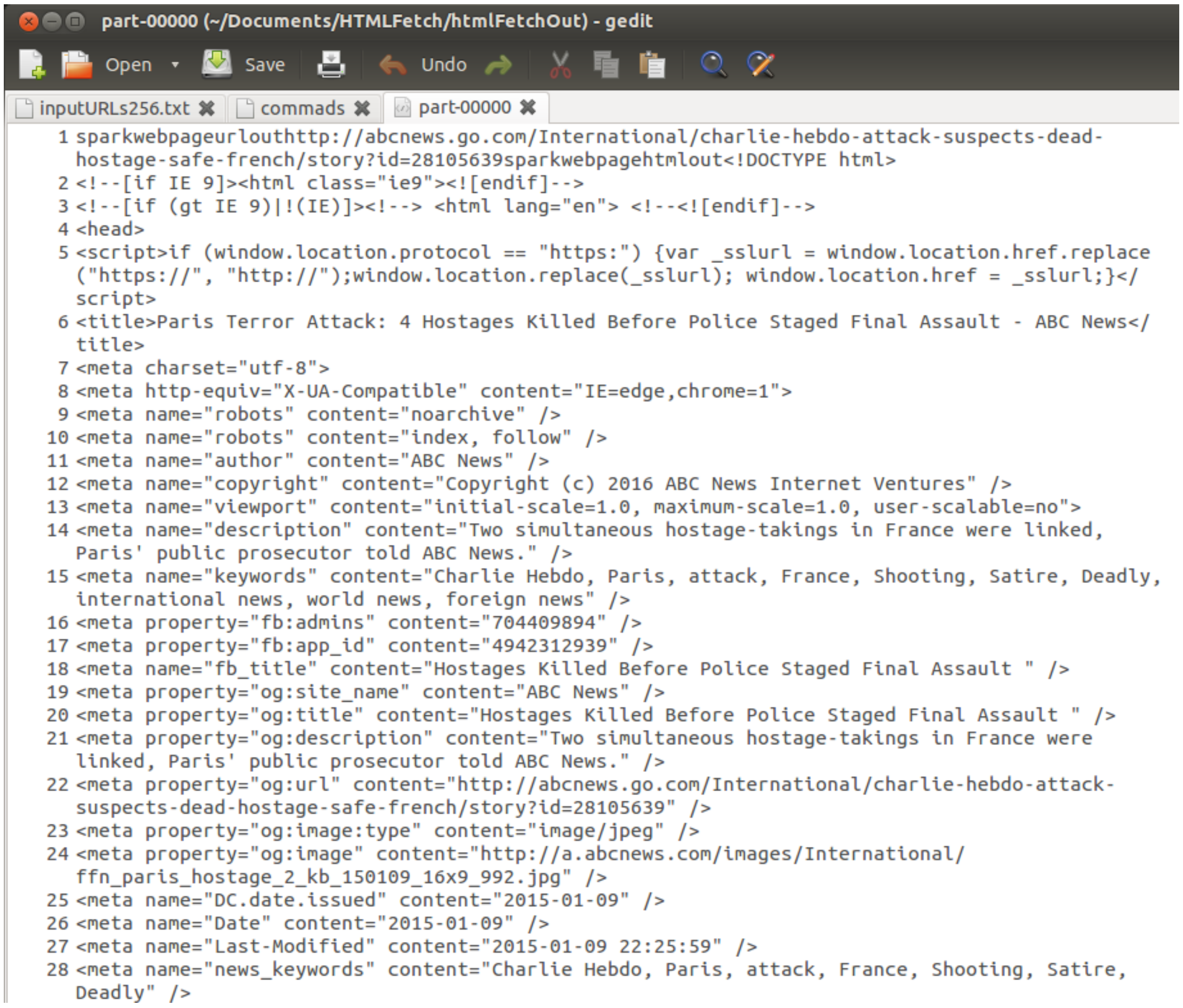
The Spark application can be built using the command:

```
sbt package
```

To run the application, use the following command:

```
spark-submit --executor-memory 1G --driver-memory 1G target/scala-2.10/htmlfetch_2.10-1.0.jar inputURLs.txt
```

This command runs the application locally (not distributed) and allocates 1G of memory for the driver. The output is stored in a folder named htmlFetchOut in the home directory of HDFS. A screenshot of sample output is shown



```
sparkwebpageurlouthttp://abcnews.go.com/International/charlie-hebdo-attack-suspects-dead-hostage-safe-french/story?id=28105639sparkwebpagehtmlout<!DOCTYPE html>
<!--[if IE 9]><html class="ie9"><![endif]-->
<!--[if (gt IE 9)|!(IE)]><!--> <html lang="en"> <!--<![endif]-->
<head>
<script>if (window.location.protocol == "https:") {var _sslurl = window.location.href.replace("https://", "http://");window.location.replace(_sslurl); window.location.href = _sslurl;}</script>
<title>Paris Terror Attack: 4 Hostages Killed Before Police Staged Final Assault - ABC News</title>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
<meta name="robots" content="noarchive" />
<meta name="robots" content="index, follow" />
<meta name="author" content="ABC News" />
<meta name="copyright" content="Copyright (c) 2016 ABC News Internet Ventures" />
<meta name="viewport" content="initial-scale=1.0, maximum-scale=1.0, user-scalable=no">
<meta name="description" content="Two simultaneous hostage-takings in France were linked, Paris' public prosecutor told ABC News." />
<meta name="keywords" content="Charlie Hebdo, Paris, attack, France, Shooting, Satire, Deadly, international news, world news, foreign news" />
<meta property="fb:admins" content="704409894" />
<meta property="fb:app_id" content="4942312939" />
<meta name="fb_title" content="Hostages Killed Before Police Staged Final Assault " />
<meta property="og:site_name" content="ABC News" />
<meta property="og:title" content="Hostages Killed Before Police Staged Final Assault " />
<meta property="og:description" content="Two simultaneous hostage-takings in France were linked, Paris' public prosecutor told ABC News." />
<meta property="og:url" content="http://abcnews.go.com/International/charlie-hebdo-attack-suspects-dead-hostage-safe-french/story?id=28105639" />
<meta property="og:image:type" content="image/jpeg" />
<meta property="og:image" content="http://a.abcnews.com/images/International/ffn_paris_hostage_2_kb_150109_16x9_992.jpg" />
<meta name="DC.date.issued" content="2015-01-09" />
<meta name="Date" content="2015-01-09" />
<meta name="Last-Modified" content="2015-01-09 22:25:59" />
<meta name="news_keywords" content="Charlie Hebdo, Paris, attack, France, Shooting, Satire, Deadly" />
```

Figure 23: HTML Fetching Output

7.4 WARC Generation

The WARC generation application was implemented as a Python script, and only requires an installation of Python; thus, it can run on the head node of the DLRL cluster. The Python script takes the same input as the HTML Fetching application described in section 7.5.

The script can be run using the following command:

```
python wgetWarc.py inputURLs.txt
```

The output WARC files are saved to a folder named warcOut.

8. Developer's Manual

The following table describes the set of tasks implied by the rest of this report and which group member is responsible.

8.1 Task Assignment Table

Table 5: Task Assignment

Team member responsible	**Task description**	**Input to task**	**Output of task**	**Tools involved**
Chris	Load Pranav's collections into HBase (Locate, extract schema information, store in HBase)	Pranav's collections of classified shooting web pages (format unknown)	Column families in HBase populated with Pranav's collections	Python/Pig [17] - extract schema information Pig - load results into HBase
	Load Mohamed's focused crawled collections into HBase (Locate, extract schema information, store in HBase)	Focused crawled collection of web pages (format consists of textual content of each web page, and any URLs contained in the web page)	Column families in HBase populated with Mohamed's collections	Pig - load textual information and URLs into HBase
	Create workflow for downloading collections from the Internet Archive	Collections of WARC files hosted on archive-it.org	WARC files stored on the cluster	Unknown - Contact Mohamed to see if this has been done before
	Create workflow for storing WARC file information into HBase	WARC files stored on HBase	Populated HBase tables	Warcbase [16] Others?
Weigang	Short URLs expanding	Short URLs from HBase	Long URLs reload into HBase	Python package: urllib, urllib2, urlparse, httplib. Java package: org.apache.pig.backend.hadoop.hbase
	URLs duplication elimination	Long URLs from HBase	If it has been fetched yet or	Java package: pig.

			not?	Own algorithm maybe.
	Fetching the website from the URLs obtained from the CMT group and create the WARC files	Long URLs from HBase	WARC files associated with the long URLs	. And Java package: pig (may also needed)
	Textual clean up for the raw WARC files directly obtain from the URLs	Raw WARC files	Cleaning WARC files	Python package: Beautifulsoup
	Timestamp added	Raw WARC files?	Timestamps of the corresponding webpage	MySQL?
Tung	Tasks related to events focused crawlers (EFC). These include getting Mohamed's EFC source code, configure, setup, modify, and eventually make it run incrementally on the server efc2.	EFC source code, efc2 server, selected seed URLs, Mohamed's dissertation about focused crawling	Relevant extracted URLs, cleaned and classified webpages, possibly in WARC format, loaded in HBase	HBase, Java, Python, Warcbase, Pig, MySQL, warc,

8.2 Extending HTML Fetching

The HTML fetching application can be extended in two ways: modifying the application to read and write directly to HBase, and checking a new proposed column under the "clean-tweet" column family when running the code incrementally.

An example of using the HBase libraries provided by Spark can be found at the following URL: https://www.mapr.com/developercentral/code/loading-hbase-tables-spark#.WEo_JLIrLmg

This example loads the HBase table as an RDD, however, the CLA team this semester found bugs with this method when running it on the DLRL cluster. Saurabh from the CLA team had this to say on the matter:

Problem description - Reads from HBase run into problems when the result set (tweets) numbers in the millions. The problems stem from the fact that a given Spark node can only load a limited number of tweets in memory. The classification team ran into problems when it was loading all the tweets from collection# 400, as it had more than 4 million tweets. The driver node would crash or bail out with an OutOfMemory exception.

Solution - Best practices suggest that a blocked read be performed on HBase, and all the records that are read as part of the blocked read should then be further processed in a parallel fashion to perform the specific tasks as part of the team's project goals. Also, the classification team set the block size to 5000 and the cache size to 1 for their actual experiments. The block size corresponds to number of rows to be read, and cache size corresponds to the number of columns read for each row. Please feel free to change this as per your requirement since the classification team needed to just read the raw tweet. Keep in mind that once you have read a block, you should make the call to get the next batch of results before 60 seconds, otherwise the scanner times out and releases the handle to HBase. So, tuning the block and cache size to best fit your requirements is important. It is also recommended that a block of data, once retrieved, is immediately cached and repartitioned across the cluster before calling any action method on the retrieved data. This results in better runtime performance as parallelism across the cluster comes into effect.

The second point of extension, running the fetcher incrementally, can be accomplished by passing an additional argument to the application corresponding to the number of webpages to fetch. The application should then scan HBase for URLs whose corresponding “fetched” column has a binary value of 0, add the URL to the input list, and mark the “fetched” field as 1, and repeat this process until the input number of webpages has been reached. If the Spark-HBase connection cannot be fixed, this would have to be implemented as a separate script that prepares an input file to the HTML Fetch application.

8.3 WARC Generation

The WARC generation script uses wget, a GNU package for retrieving web resources, to generate WARC files. The script calls wget as a shell command, thus wget needs to be installed.

The wget command used by the script is as follows:

```
wget --mirror --warc-file=out --html-extension -PwarcOut --convert-links  -t 1 url
```

--mirror mirrors all resources required to serve the webpage as opposed to just generating a WARC file for the index.html file.

--warc-file=out turns on WARC output and generates a WARC file with the specified file name

--html-extension appends “.html” to any HTML files downloaded

-P specifies the output directory, in this case a folder named “warcOut”

--convert-links changes the links in downloaded files to point to their corresponding local file

-t specifies the number of times to try a connection

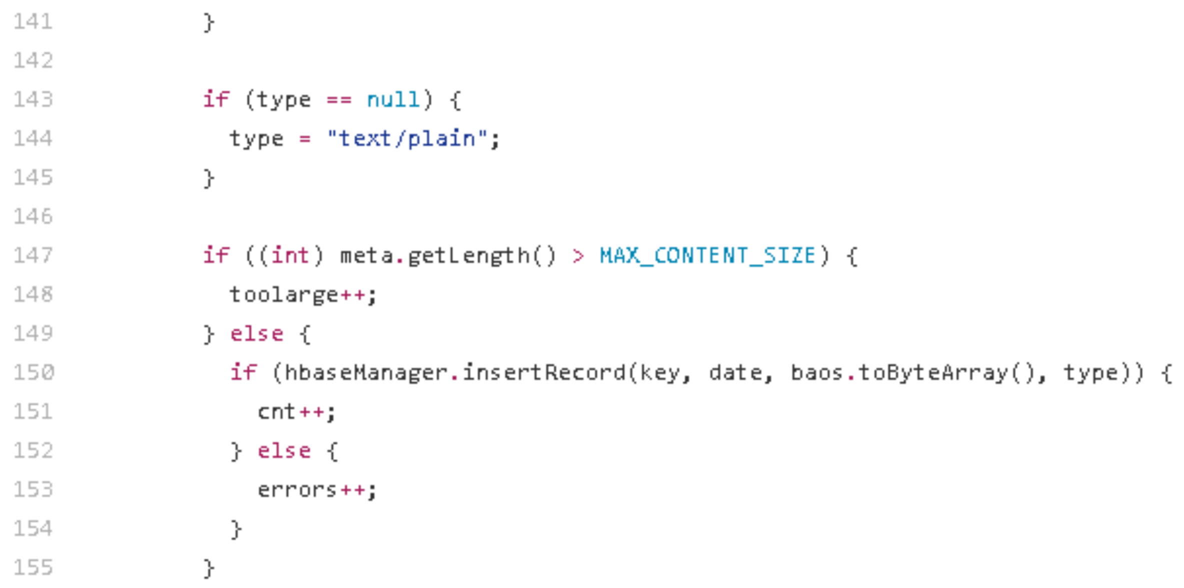
The command argument “url” is the URL of the webpage.

8.4 WARC Ingestion

The version of warcbase modified by Andrej Galad can be found at the following URL:
https://github.com/VTUL/warcbase

The screenshot shown below in Figure 24 is of a file in the repository in the location:
warcbase/warcbase-hbase/src/main/java/org/warcbase/ingest/IngestFiles.java

Line 150 inserts the record into the HBase table. It is expected by the application that the HBase table has a column for a key, date, byte array, and type. The key is generated by a function that takes the URL of the webpage as input. The byte array is the corresponding bytes of an input WARC file. In extending this work, the application could be modified to take a schema file, and extract the corresponding information from the WARC file as opposed to writing its output as a byte array.



```
141        }
142
143        if (type == null) {
144          type = "text/plain";
145        }
146
147        if ((int) meta.getLength() > MAX_CONTENT_SIZE) {
148          toolarge++;
149        } else {
150          if (hbaseManager.insertRecord(key, date, baos.toByteArray(), type)) {
151            cnt++;
152          } else {
153            errors++;
154          }
155        }
```

Figure 24: warcbase Storage

9. Acknowledge

We would like to thank Dr. Edward Fox not only for giving us an amazing opportunity to take this wonderful course, but also for his extremely helpful and insightful comments, and encouraging and continuous support during the class. In addition, we are grateful to Mr. Mohamed Magdy, Mr. Sunshin Lee, and Mr. Islam Harb for their valuable suggestions and technical supports throughout the project. Mr. Magdy not only gave us access to his event focussed crawler but also helped resolve technical issues. Mr. Lee provided us with very useful advices. Mr. Harb helped us resolve technical issues with setting up the computing environment of the ef2 server for the crawler. We are also thankful to the other teams, CLA, CMT, CTA, FE and SOLR for their comments and collaborations. Finally, thanks go to NSF for support through grants IIS-1319578 and 1619028.

9. References

[1] Mohamed Magdy Gharib Farag. 2016. *Intelligent Event Focused Crawling.* Virginia Tech, Blacksburg, VA, USA. http://hdl.handle.net/10919/73035 (*last accessed 10/11/2016*)

[2] Yufeng Ma, and Dong Nan. 2016. *Collection Management for IDEAL*. Virginia Tech, Blacksburg, VA, USA. http://hdl.handle.net/10919/70930 (*last accessed 10/11/2016*)

[3] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

[4] The Apache Software Foundation. *HBase*. http://hbase.apache.org/ (*last accessed 10/11/2016*)

[5] Events Archiving Collections. http://www.eventsarchive.org/node/18 (*last accessed 10/11/2016*)

[6] Archive-It Built at the Internet Archive. https://www.archive-it.org/ (*last accessed 10/11/2016*)

[7] SteveKallestad. 2014-02-21. *Hadoop Distributed File System*. https://wiki.apache.org/hadoop/HDFS (*last accessed 10/11/2016*)

[8] The Apache Software Foundation. *Hadoop*. http://hadoop.apache.org/ (*last accessed 10/11/2016*)

[9] Wikipedia. *Web Crawler.* https://en.wikipedia.org/wiki/Web_crawler (*last accessed 10/11/2016*)
Creative Commons Attribution-ShareAlike License.

[10] Events Archiving Facilities: DLRL Hadoop Cluster. Virginia Tech, Blacksburg, VA, USA. http://www.eventsarchive.org/node/12 (*last accessed 10/11/2016*)

[11] Jonathan Hedley 2009-2016. *jsoup HTML parser*. https://jsoup.org/ (*last accessed 10/11/2016*)

[12] Internet Archive. 2003-2011. *About Heritrix*. http://crawler.archive.org/index.html (*last accessed 10/11/2016*)

[13] Wikipedia. *Web ARChive*. https://en.wikipedia.org/wiki/Web_ARChive (*last accessed 10/11/2016*)
Creative Commons Attribution-ShareAlike License.

[14] Oracle Corporation. *MySQL*. https://www.mysql.com/ (*last accessed 10/11/2016*)

[15] Program Creek. *How to make a Web crawler using Java.* 2008.
http://www.programcreek.com/2012/12/how-to-make-a-web-crawler-using-java/ (*last accessed 10/11/2016*)

[17] The Apache Software Foundation. *Pig.* https://pig.apache.org/ (*last accessed 10/11/2016*)

[18] Andrej Galad. 2016. *VTUL/warcabse.* Virginia Tech, Blacksburg, VA, USA.
https://github.com/VTUL/warcbase (*last accessed 10/11/2016*)

[19] Stephen Merity. April 2, 2014. *Navigating the WARC file format.* Common Crawl.
http://commoncrawl.org/2014/navigating-the-warc-file-format (*last accessed 10/11/2016*)