A Appendix I

The following script flags all webpages that include one or more mentions of the term 'climate change' and stores the full text of those captures. We begin with an overview of the process of running jobs on the cluster, and then provide specific code. For questions, please contact the authors.

A.1 Overview

Running scripts on the cluster requires a basic understanding of bash (Unix) shell commands using the Command Line on a home computer (on a Mac, this is the program "Terminal"). For a basic run down of bash commands, see `http://cli.learncodethehardway.org/bash_cheat_sheet.pdf`.

Begin by opening a bash shell on a home desktop, and using an ssh key obtained from Altiscale to log in. Once logged in, you will be on your personal workbench and now have to use a script editor (such as Vi `http://www.catonmat.net/download/bash-vi-editing-mode-cheat-sheet.pdf`). Come up with a name for the script, open the editor, and then either paste or write the desired script in the editor, close and save the file (to your personal workbench on the cluster).

Scripts must be written in Hadoop-accesable languages, such as Apache Pig, Hive, Giraph or Oozie. Apache langagues are SQL-like, which means if you have experience with SQL, MySQL, SQLlite or PostgreSQL (or R or Python), the jump should not be too big. For text processing, Apache Pig is most appropriate, whereas for link anaylsis, Hive is best. The script below is written in Apache Pig and a manual can be found at `https://pig.apache.org/`. For an example of some scripts written for this cluster, see `https://webarchive.jira.com/wiki/display/Iresearch/IA+-+GOV+dataset+-+Altiscale`. May be easiest to it "clone" the "archive analysis" file hosted on GitHub from Vinay Goel `https://github.com/vinaygoel` or three basic scripts from Emily Gade `https://github.com/ekgade/.govDataAnalysis` and use those as a launchpoint. If you don't know how to use GitHub, see here: `https://guides.github.com/activities/hello-world/`(it is actually quite straightforward).

Because Apache languages have limited functionality, users may want to write user defined functions in a program like Python. A tutorial about how to do this can be found at `https://help.mortardata.com/technologies/pig/writing_python_udfs`.

Once a script is written, you will want to run it on a segment of the cluster. This requires another set of Unix style Hadoop shell commands (see `http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html`). Users must then specify the file path(s), the desired output directory, and where the script can be found.

A.2 Getting a Key

As discussed above, this script is run from your workbench on the cluster. To gain acesss, you will need to set up an SSH "key" with Altiscale (see `http://documentation.altiscale.`

com/configure-ssh-from-mac-linux). Once you have obtained and sent your SSH key to Alitscale, you can log in using any bash shell from your desktop with the commmand "ssh altiscale".

A.3 Locating the Data

The Altiscale cluster houses 9 "buckets" of .GOV data. Each bucket contains hundreds or thousands of Web Archive Files (older version are "ARC" files, newer version are "WARC" files, but they have all the same fields). Each WARC/ARC file contains captures from the same crawl, but it (a) won't contain all of the captures from a given crawl, and (b) since the crawl is doing a lot of things simultaneously, captures of a single site can be located in diffrerent WARC files.

With so much data, there is no simple "table" or directory that can be consulted to locate a specific web page. The best way to find specific pages is to use Hive to query the CDX database. See Vinay Goel's GitHub for details about how to query CDX: `https://github.com/vinaygoel/archive-analysis/tree/master/hive/cdx`. If a user know exactly what he or she wants (all the captures of the whitehouse.gov mainpage, or all the captures from September 11, 2001), the CDX can tell you where to find them. Otherwise, users will want to query all of the buckets because there is no easy way to learn where results are stored. (Though we advise first testing scripts on a single bucket or WARC file.)

First, use the command line with SSH interface to query the data directories and see which buckets or files to run a job over. This requires the Hadoop syntax to "talk" to the cluster where all the data is stored. The cluster has a user-specific directory where users can store the results of scrapes. A user's local work bench does not have enough space to save them.

Whenever users "talk" from a user's local workbench to the main cluster, users need to use 'hadoop fs -' and then the bash shell command of interest. For a list of Hadoop-friendly bash shell comands, see: `http://hadoop.apache.org/docs/current1/file_system_shell.html`. For example, the line of code

```
hadoop fs -ls
```

pulls a listing of the files in your personal saved portion of the cluster (in addition to the local workbench, each user has a file directory to save the results). As well,

```
hadoop fs -ls /dataset-derived/gov/parsed/arcs/bucket-2/
```

would draw up all the files in Bucket 2 of the parsed text ARCS directory.

A.4 Defining Search Terms

Scripts that deal with text are best written in Apache Pig. Hadoop also supports Apache Hive, Giraffe and Spark. To find and collect terms or URLs of interest, users will need to write a script. For example, users might write a script to flag any captures that have a mention of a global warming term, and return the date of the capture, URL, page title, checksum, and the parsed text. This script is saved on your local workbench and needs to have a .pig suffix. Users will need to use some sort of bash editor to write and store the script such as vi (details about

how to use vi can be found above). Script is below. The first four lines are defaults and also set the memory.

Script begins:

```
SET default_parallel 100;
SET mapreduce.map.memory.mb 8192;
SET mapred.max.map.failures.percent 10;
REGISTER lib/ia-porky-jar-with-dependencies.jar;
DEFINE FROMJSON org.archive.porky.FromJSON();
DEFINE SequenceFileLoader org.archive.porky.SequenceFileLoader();
DEFINE SURTURL org.archive.porky.SurtUrlKey();
```

The sequence file loader pulls the files out of the ARC/WARC format and makes them readable. Note, when they were put into the ARC/WARC format, the were run through a HTML parser to remove the HTML boilerplate. However, if the file was not in HTML to begin with, the parser will just produce symbols and this won't fix it. Users will have to deal with those issues separately.

When loading data on the command line (instructions below), give the data a name (here $I_Parsed_Data) and make sure to use the same "name" for the data in the command line command. This is a stand-in for the name of the directory or file over which you will run a script.

```
Archive = LOAD "\$I_PARSED_DATA" USING SequenceFileLoader()
AS (key:chararray, value:chararray);
Archive = FOREACH Archive GENERATE FROMJSON(value) AS m:[];
Archive = FILTER Archive BY m#‘errorMessage’ is null;
ExtractedCounts = FOREACH Archive GENERATE m#‘url’ AS src:chararray,
    SURTURL(m#‘url’) AS surt:chararray,
    REPLACE(m#‘digest’,‘sha1:’,’’) AS checksum:chararray,
    SUBSTRING(m#‘date’, 0, 8) AS date:chararray,
    REPLACE(m#‘code’, ‘[^\\p{Graph}]’, ‘ ’) AS code:chararray,
    REPLACE(m#‘title’, ‘[^\\p{Graph}]’, ‘ ’) AS title:chararray,
    REPLACE(m#‘description’, ‘[^\\p{Graph}]’, ‘ ’)AS description:chararray,
    REPLACE(m#‘content’, ‘[^\\p{Graph}]’, ‘ ’) AS content:chararray;
```

The above code block says: for each value and key pair, pull out the following fields. Chararray means character array - so a list of characters with no limits on what sort of content may be included in that field. The next line selects the first eight characters of the date string (year, month, day). The full format is year, month, day, hour, second. Unicode errors can wreck havoc on script and outputs. The regular expression pGraph means "all printed characters"– e.g., NOT new lines, carriage returns, etc. So, this query finds anything that is not text, punctuation and white space, and replaces it with a space. Also note that because Pig is under-written in Java, users need two escape characters in these scripts (whereas only one is needed in Python).

```
UniqueCaptures = FILTER ExtractedCounts BY content
```

```
MATCHES '.*natural\\s+disaster.*' OR content MATCHES
'.*desertification.*' OR content MATCHES
'.*climate\\s+change.*' OR content MATCHES
'.*pollution.*' OR content MATCHES
'.*ocean\\s+acidification.*' OR content MATCHES
'.*anthropocene.*' OR content MATCHES
'.*anthropogenic.*' OR content MATCHES
'.*greenhouse\\s+gas.*' OR content MATCHES
'.*climategate.*' OR content MATCHES
'.*climatic\\s+research\\s+unit.*' OR content MATCHES
'.*security\\s+of\\s+food.*' OR content MATCHES
'.*global\\s+warming.*' OR content MATCHES
'.*fresh\\s+water.*' OR content MATCHES
'.*forest\\s+conservation.*' OR content MATCHES
'.*food\\s+security.*';
```

This filters out the pages with key words of interest (in this case words related to climate change) and keeps only those pages.

```
STORE UniqueCaptures INTO '\$O_DATA_DIR' USING PigStorage('\u0001');\\
```

This stores the counts the file name given to it. The "using pigstorage" function allows users to set their own delimiters. I chose a Unicode delimiter because commas/tabs show up in the existing text. And, since I stripped out all Unicode above, this should be clearly a new field.

Save this script to your local workbench.

Another option would be to count all the mentions of specific terms. Instead of the above, users would run:

```
SET default_parallel 100;
SET mapreduce.map.memory.mb 8192;
SET mapred.max.map.failures.percent 10;
REGISTER lib/ia-porky-jar-with-dependencies.jar;
```

This line allows you to load user defined fuctions from a Python file:

```
REGISTER 'UDFs.py' USING jython AS myfuncs;
DEFINE FROMJSON org.archive.porky.FromJSON();

DEFINE SequenceFileLoader org.archive.porky.SequenceFileLoader();
DEFINE SURTURL org.archive.porky.SurtUrlKey();
Archive = LOAD '\$I\_PARSED\_DATA' USING SequenceFileLoader()
AS (key:chararray, value:chararray);
Archive = FOREACH Archive GENERATE FROMJSON(value) AS m:[];
Archive = FILTER Archive BY m#'errorMessage' is null;
ExtractedCounts = FOREACH Archive GENERATE m#'url' AS src:chararray,
    SURTURL(m#'url') AS surt:chararray,
    REPLACE(m#'digest','sha1:','') AS checksum:chararray,
```

```
    SUBSTRING(m#‘date’, 0, 8) AS date:chararray,
    REPLACE(m#‘code’, ‘[^\\p{Graph}]’, ‘ ’) AS code:chararray,
    REPLACE(m#‘title’, ‘[^\\p{Graph}]’, ‘ ’) AS title:chararray,
    REPLACE(m#‘description’, ‘[^\\p{Graph}]’, ‘ ’)AS description:chararray,
    REPLACE(m#‘content’, ‘[^\\p{Graph}]’, ‘ ’) AS content:chararray;
```

If a user has function which selects certain URLs of interest and groups all other URLs as “other”, they would run it only on the URL field. And, if a user has a function that collects words of interest and counts them as well as total words, the user should run that through the content field. Code for using those UDFs would look something like this:

```
UniqueCaptures = FOREACH ExtractedCounts GENERATE myfuncs.pickURLs(src),
    src AS src,
    surt AS surt,
    checksum AS checksum,
    date AS date,
    myfuncs.Threat_countWords(content);
```

In Pig, and the default delimiter is ‘’ (new line) but many ‘’ appear in text. So one must get rid of all the new lines in the text. This will affect our ablity to do text parsing by paragraph, but sentences will still be possible. Code to get rid of the ‘’ (new line deliminaters) which are causing problems with reading in tables might look something like this:

```
UniqueCaptures = FOREACH UniqueCaptures GENERATE REPLACE(content, ‘\n’, ‘ ’);
```

To get TOTAL number of counts of webpages, rather than simply unique observations, merge with checksum data:

```
Checksum = LOAD ‘\$I\_CHECKSUM\_DATA’ USING PigStorage() AS (surt:chararray,
date:chararray, checksum:chararray);

CountsJoinChecksum = JOIN UniqueCaptures BY (surt,
checksum), Checksum BY (surt, checksum);

FullCounts = FOREACH CountsJoinChecksum GENERATE
    UniqueCaptures::src as src,
    Checksum::date as date,
    UniqueCaptures::counts as counts,
    UniqueCaptures::URLs as URLs;
```

This would sort counts by original “source” or URL:

```
GroupedCounts = GROUP FullCounts BY src;
```

This fills in the missing counts and stores results:

```
GroupedCounts = FOREACH GroupedCounts GENERATE
    group AS src,
    FLATTEN(myfuncs.fillInCounts(FullCounts)) AS (year:int,
    month:int, word:chararray, count:int, filled:int,
    afterlast:int, URLs:chararray);
STORE GroupedCounts INTO ‘\$O\_DATA\_DIR’;
```

The UDFs mention here (pickURLs, Threat_countWords, and FillinCounts) are written in Python and can be seen in at the bottom of this Appendix.

A.5 Running the Script

To run this script, type the following code into the command line, after having logged in the Altiscale cluster with your ssh key. Users will select the file or bucket they want to run the script over, and type in an "output" directory (this will appear on your home/saved data on the cluster, not on your local workbench). Finally, users need to tell Hadoop which script they want to run. The I_PARSED_DATA was defined as the location of the data to run the script over in the script above. Here we telling the computer that this bucket is the I_PARSED_DATA. Next, one must load the CHECKSUM data, and finally, give the output directory, and the location of your script.

The following should be run all as one line:

```
pig -p I_PARSED_DATA=/dataset-derived/gov/parsed/arcs/bucket-2/
-p I_CHECKSUM_DATA=/dataset/gov/url-ts-checksum/
-p O_DATA_DIR=place_where_you_want_the_file_to_end_up
location_of_your_script/scriptname.pig
```

A.6 Exporting Results

Lastly, to remove results from the cluster users need to open a new Unix shell on their local machine that is NOT logged in to the cluster with their ssh key. Then type the location of the file they'd like to copy and give it a file path for where they'd like to put it on their desktop. For example:

The following should be run all as one line:

```
scp -r altiscale:~/results_location
/location_on_your_computer_you_want_to_move_results_to/
```

For additional scripts and for those with programing experience, see Vinay Goel's GitHub at `https://github.com/vinaygoel/archive-analysis`. For stepwise instruction of a wordcount script, see Emily Gade's GitHub at `https://github.com/ekgade/.govDataAnalysis`.

Python UDFs:

```
#import packages
from collections import defaultdict
import sys
import re

#define output schema so the UDF can talk to Pig
@outputSchema("URLs:chararray")

# define Function
def pickURLs(url):
   try:
      # these can be any regular expressions
      keyURLs = [
      'state\.gov',
```

```
      'treasury\.gov',
      'defense\.gov',
      'dod\.gov',
      'usdoj\.gov',
      'doi\.gov',
      'usda\.gov',
      'commerce\.gov',
      'dol\.gov',
      'hhs\.gov',
      'dot\.gov',
      'energy\.gov',
      'ed\.gov',
      'va\.gov',
      'dhs\.gov',
      'whitehouse\.gov',
      '\.senate\.gov',
      '\.house\.gov']

      URLs = []
      for i in range(len(keyURLs)):
         tmp = len(re.findall(keyURLs[i], url, re.IGNORECASE))
         if tmp > 0:
            return keyURLs[i]
      return 'other'

# counting words

#define output schema as a "bag" with the word and then the count of the word
@outputSchema("counts:bag{tuple(word:chararray,count:int)}")

def Threat_countWords(content):
   try:
      # these can be any regular expressions
      Threat_Words = [
            '(natural\sdisaster)',
            '(global\swarming)',
            '(fresh\swater)',
            '(forest\sconservation)',
            '(food\ssecurity)',
            '(security\sof\sfood)',
            'desertification',
            '(intergovernmental\spanel\son\sclimate\schange)',
            '(climatic\sresearch\sunit)',
            'climategate',
            '(greenhouse\sgas)',
            'anthropogenic',
            'anthropocene',
            '(ocean\sacidification)',
```

```
            'pollution',
            '(climate\schange)']

#if you want a total of each URL or page, include a total count
      threat_counts  = defaultdict(int)
      threat_counts['total'] = 0

      if not content or not isinstance(content, unicode):
         return [(('total'), 0)]
      threat_counts['total'] = len(content.split())

      for i in range(len(Threat_Words)):
         tmp = len(re.findall(Threat_Words[i], content, re.IGNORECASE))
         if tmp > 0:
            threat_counts[Threat_Words[i]] = tmp

      # Convert counts to bag
      countBag = []
      for word in threat_counts.keys():
         countBag.append( (word, threat_counts[word] ) )
      return countBag

## filling in counts using CHECKSUM and carrying over counts
from the "last seen" count

@outputSchema("counts:bag{tuple(year:int, month:int, word:chararray, count:int,
filled:int, afterLast:int, URLs:chararray)}")

def fillInCounts(data):
   try:
      outBag = []
      firstYear = 2013
      firstMonth = 9
      lastYear = 0
      lastMonth = 0
 # used to compute averages for months with multiple captures
 # word -> (year, month) -> count
      counts = defaultdict(lambda : defaultdict(list))
      lastCaptureOfMonth = defaultdict(int)
      endOfMonthCounts = defaultdict(lambda : defaultdict(lambda:
      dict({'date':0,'count':0})))
      seenDates = {}
#ask for max observed date
      for (src, date, wordCounts, urls) in data:
         for (word, countTmp) in wordCounts:
            year = int(date[0:4])
            month = int(date[4:6])
            if isinstance(countTmp,str) or isinstance(countTmp,int):
```

```
            count = int(countTmp)
         else:
            continue

         ymtup = (year, month)
         counts[word][ymtup].append(count)

         if date > lastCaptureOfMonth[ymtup]:
            lastCaptureOfMonth[ymtup] = date
         if date > endOfMonthCounts[word][ymtup]['date']:
            endOfMonthCounts[word][ymtup]['date'] = date
            endOfMonthCounts[word][ymtup]['count'] = count

         seenDates[(year,month)] = True

         if year < firstYear:
            firstYear = year
            firstMonth = month
         elif year == firstYear and month < firstMonth:
            firstMonth = month
         elif year > lastYear:
            lastYear = year
            lastMonth = month
         elif year == lastYear and month > lastMonth:
            lastMonth = month

   for word in counts.keys():
# The data was collected until Sep 2013
make sure that you aren't continueing into the future
      years = range(firstYear, 2014)
      useCount = 0
      afterLast = False
      filled = False
      ymLastUsed = (0,0)
      for y in years:
         if y > lastYear:
            afterLast = True
         if y == firstYear:
            mStart = firstMonth
         else:
            mStart = 1
         if y == 2013:
            mEnd = 9
         else:
            mEnd = 12
         for m in range(mStart, mEnd+1):
            if y == lastYear and m > lastMonth:
               pass
```

```
            if (y,m) in seenDates:

# Output sum, as we will divide by sum of totals later
               useCount = sum(counts[word][(y,m)])
               ymLastUsed = (y,m)
               filled = False
            else:
# If we didn't see this date in the capture, we want to use the last capture we saw
# previously (we might have two captures in Feb, so for Feb we output both,
# but to fill-in for March we would only output the final Feb count)
# Automatically output an assumed total for each month (other words
# may no longer exist)

               if endOfMonthCounts[word][ymLastUsed]['date'] ==
               lastCaptureOfMonth[ymLastUsed]:
                  useCount = endOfMonthCounts[word][ymLastUsed]['count']
               else:
                  continue
               filled = True
            if useCount == 0:
               continue
            outBag.append((y, m, word, useCount, int(filled), int(afterLast), urls))
```

B Appendix II: Lists of URLs and Terms

Figure 8: URLS

Financial	Terrorism	Climate
cftc\.gov	atf\.gov	doe\.gov
doj\.gov	cdc\.gov	doi\.gov
fanniemae\.com	cia\.gov	dot\.gov
fasb\.org	defense\.gov	eia\.gov
fdic\.gov	dod\.gov	epa\.gov
federalreserve\.gov	doe\.gov	fema\.gov
ffiec\.gov	doj\.gov	fws\.gov
fhfa\.gov	dot\.gov	gao\.gov
fhfb\.gov	eia\.gov	gsa\.gov
finra\.org	faa\.gov	nasa\.gov
freddiemac\.com	fbi\.gov	noaa\.gov
fslic	fema\.gov	nps\.gov
ftc\.gov	gao\.gov	nsf\.gov
gao\.gov	gsa\.gov	occ\.gov
ginniemae\.gov	ice\.gov	state\.gov
gsa\.gov	nsa\.gov	usaid\.gov
hhs\.gov	nsf\.gov	usda\.gov
homeloans\.va\.gov	secretservice\.gov	usgs\.gov
makinghomeaffordable\.	state\.gov	
ncua\.gov	treasury\.gov	
sec\.gov	usaid\.gov	
sipc\.org	usda\.gov	
treasury\.gov	usphs\.gov	

Figure 9: Terrorism Terms

(9//11*)
(al\-qa*)
(alien\ssmuggl*)
(arms\sprolifer*)
(arms\ssmuggl*)
(arms\stransfer*)
(assassin*)
(atrocit*)
(authoritarian\spopul*)
(ballistic\smissile)
(bin\sladen)
(biological\sweapon*)
(biopreparedness)
(bioregulator*)
(biosecurity)
(bioterror*)
(border\ssecurity)
(catastrophic\shealth\seve
(chemical\sweapon*)
(collateralize*)
(conventional\sarm*)
(counterterror*)
(critical\sinfrastructure)
(cyber\-attack*)
(cyber\sattack*)
(cyberattack)
(cybersecurit*)
(cyberterror*)
(cyberwar*)
(cyberwarfare)
(dirty\sbomb)
(disease*)
(dual\-use\sgood*)
(electronic\swar*)
(fissile\smaterial)
(food\ssecurity)
(fragile\sstate*)
(fundamentalis*)
(genocide*)
(hijack*)
(hostile\sstate*)
(if\syou\ssee\ssomething\,
(illegal\smigration)
(improvised\sexplosive\sde
(insurgen*)
(irresponsible\sstate*)
(jihad)
(known\sand\ssuspected\ste
(ksts)
(mass\scasualt*)
(massive\scasualt*)
(military\sforce*)
(non\-state\sactor*)
(pandemic*)
(proliferat*)
(proliferation)
(radiological)
(securitiz*)
(security)
(september\s11*)
(suspicious\sactivity)
(taliban)
(terror*)
(terrorism)
(terrorist)
(threat*)
(violat[a-z]+?\s?o?f?\sint
(violat[a-z]+?\s?o?f?\su?n
(violent\sconflict)
(violent\sextremis*)
(weapon[a-z]+?\sof\smass\s
(wmd)
(zoonotic\sdisease*)

SELECT TERMS

(proliferat*)
(september\s11*)
(terrorism)
(terrorist)
(weapon[a-z]+?\sof\smass\s
(wmd)

Figure 10: Finance Terms

(adjustable\-rate\smortgag
(bailout*)
(bubble)
(capital\srequirement*)
(cdo)
(conservatorship)
(default)
(exposure)
(fannie\smae)
(financial\sfraud)
(foreclosure)
(freddie\smac)
(ginnie\smae)
(haircut)
(home\sprice*)
(insolvent)
(lehman)
(leverag*)
(liquidity)
(mortgage-backed)
(panic)
(plunge)
(predatory)
(receivership)
(shadow)
(sluggish\seconomic\sgrowt
(solvency)
(speculat*)
(sub\-prime)
(subprime)
(systemic\srisk)
(toxic)

SELECT TERMS
(bubble)
(default)
(fannie\smae)
(foreclosure)
(freddie\smac)
(haircut)
(lehman)
(liquidity)
(subprime)
(systemic\srisk)

Figure 11: Climate Terms

(adaptation)
(alternative\senergy)
(anthropoc*)
(anthropog*)
(carbon)
(cfc)
(clean\senergy)
(climate)
(climate\schange)
(climategate)
(co2)
(desertification)
(emission*)
(energy\sefficiency)
(fresh\swater)
(global\swarm*)
(greenhouse)
(gse)
(hockey\sstick)
(hydrocarbon*)
(ipcc)
(kyoto)
(methane)
(mitigation)
(ozone)
(sea\slevel\srise)
(sea\ssurface)
(unfcc)
(united\snations\sframewor
(warming)

SELECT TERMS
(anthropoc*)
(anthropog*)
(cfc)
(climate\schange)
(co2)
(global\swarm*)
(greenhouse)
(ipcc)
(kyoto)
(ozone)