

IDEAL Pages

VIRGINIA TECH

BLACKSBURG

CS 4624

PAUL ALY & GASPER GULOTTA

CLIENT: MOHAMED MAGDY

Table of Contents

Table of Figures	2
Executive Summary.....	3
User’s Manual	4
Necessary Software	4
Operating the Script Linearly	5
Individual Functions in Script	7
Notes	8
Developer’s Manual.....	9
Project Architecture	10
Overall IDEAL Project Architecture (1).....	10
Project Flow	11
Program Files	11
Concurrent Script	12
Hadoop Scripts	12
Hadoop Concepts (MapReduce)	13
Project Dependencies	14
Possible Expansions	14
Lessons Learned	15
Timeline.....	16
Problems/Solutions.....	18
Acknowledgements.....	21
References	23

Table of Figures

Figure 1: Screenshot of directory with populated with web archive files.	5
Figure 2: Sample Command for Script	6
Figure 3: Directory after Web Archive files have been processed	7
Figure 4: IDEAL Prproject Architecture	10
Figure 5: IDEAL Pages project flow	11
Figure 6: Hadoop Concepts.....	13
Figure 7: Initial project schedule.....	16
Figure 8: Final Project Schedule.....	17

Executive Summary

The purpose of this project was to simplify a process of unzipping web archive files, parsing them, and indexing them into a Solr instance. These tasks are only a subset of tasks that are part of a greater project called IDEAL. The main goal of the project is to build an 11-node cluster that will take roughly 10TB of webpages collected from various events, and ingest, filter, analyze, and provide convenient access to these webpages in a user interface. The IDEAL Pages portion of this project is critical to the overall success of the mission. In order to provide desired services with the data, the data needs to be successfully delivered to Solr. However when working with nearly 10TB of compressed data, it proves to be a rather difficult task. The primary objective was to create a Python script to convert all of the web archive files into raw text files containing the text found on these web pages. Through the use of multiple existing tools, and software developed in the process, the task was able to be accomplished. A tool for working with web archive files called Hanzo Warc Tools and was incorporated into the Python script to unpack the files. To extract the text from the HTML we made use of a program called BeautifulSoup which was able to create files which that could be easily indexed into Solr. A key element to making the IDEAL project timely however is to distribute this process through the use of Hadoop. Hadoop grants the ability to run the same process on multiple machines concurrently to effectively reduce the overall runtime of a task. To accomplish this task, it was necessary to split the script into multiple pieces and run them through Hadoop with the use of Map/Reduce. Using one of the IDEAL Project machines and Cloudera, it was possible to work with Hadoop and Map/Reduce. The outcome of this project resulted in an efficient way to process web archive files and remains extendable to optimize distribution of the tasks involved.

User's Manual

The purpose of the user's manual is to provide a resource for those using the script to quickly run it on their machine. Additionally, anyone who is looking to perform any of the individual tasks that were part of this project may also find this manual helpful.

The user's manual is broken up into the following parts:

- Necessary Software
- Operating the Script Linearly
- Individual Functions in Script
- Notes

Necessary Software

It is assumed that anyone attempting to run this software does so with the use of a Unix distribution. If a user wishes to work with Windows it is recommended to install PuTTY and run all commands from there.

Software/Tools:

- Terminal Access
- Python 3.3.0 (<https://www.python.org/download/releases/3.3.0>)
- Hanzo-warc-tools 0.2 (<https://pypi.python.org/pypi/hanzo-warc-tools/0.2>)
- Beautiful Soup (<http://www.crummy.com/software/BeautifulSoup/>)
- Solr Access
- processWarcDir.py (Created within this project scope and available through VTechWorks)

- An Web Archive File (.warc) to be processed

Operating the Script Linearly

The following will describe how to operate this script such that one web archive file is processed and indexed into Solr.

- 1) Initialize Solr instance. If need be, use the tutorial here:
(<https://lucene.apache.org/solr/tutorial.html>)
- 2) After all of the software is installed ensure that the processWarcDir.py is in the same directory as the web archive file and hanzo-warc-tools. Below shows a screenshot of an example directory before execution of the script.

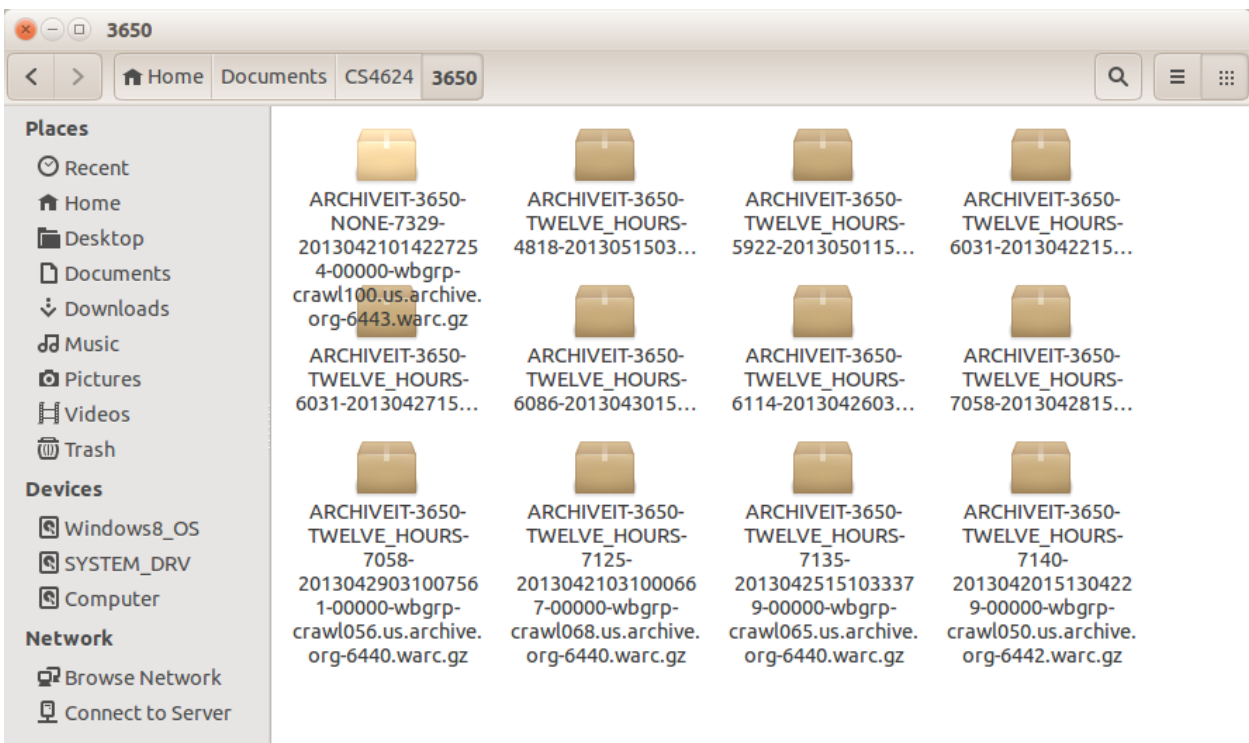
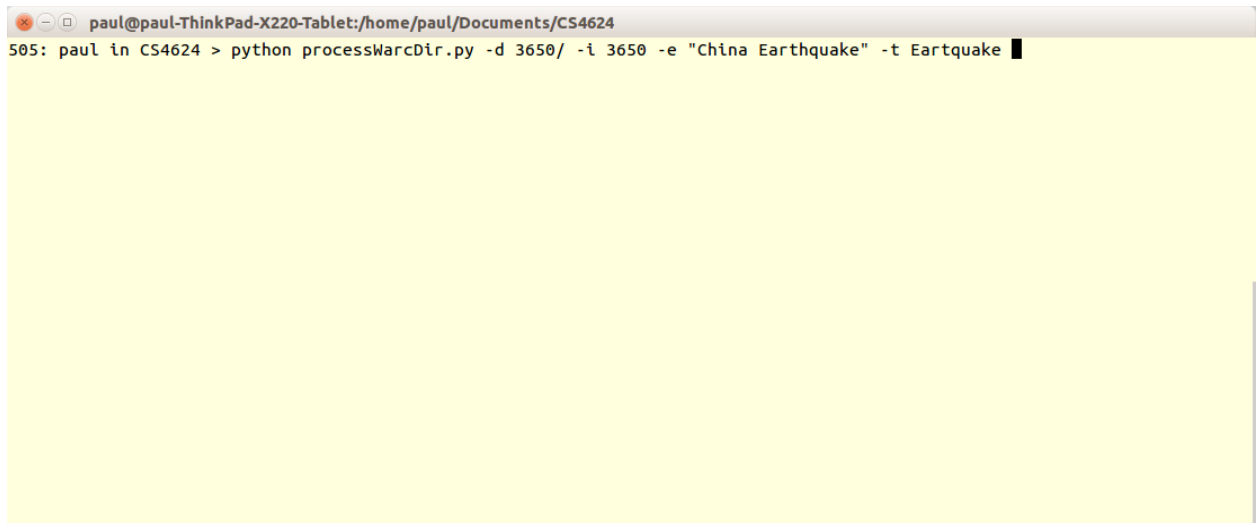


Figure 1: Screenshot of directory with populated with web archive files. Figure X: Screenshot of directory with populated with web archive files.

3) Navigate to the working directory and run the program using the following format:

```
$ python processWarcDir.py -d <working directory> -e "Event Name" -t <Event Type>
```

A terminal window with a yellow background. The title bar reads "paul@paul-ThinkPad-X220-Tablet:/home/paul/Documents/CS4624". The terminal text shows the command: "505: paul in CS4624 > python processWarcDir.py -d 3650/ -i 3650 -e "China Earthquake" -t Eartquake".

```
paul@paul-ThinkPad-X220-Tablet:/home/paul/Documents/CS4624
505: paul in CS4624 > python processWarcDir.py -d 3650/ -i 3650 -e "China Earthquake" -t Eartquake
```

Figure 2: Sample Command for Script

4) After completion of the script, all html files will be placed into either the HTTP folder or HTTPS folder and the text files containing information on each file will be produced as is shown in Figure X.

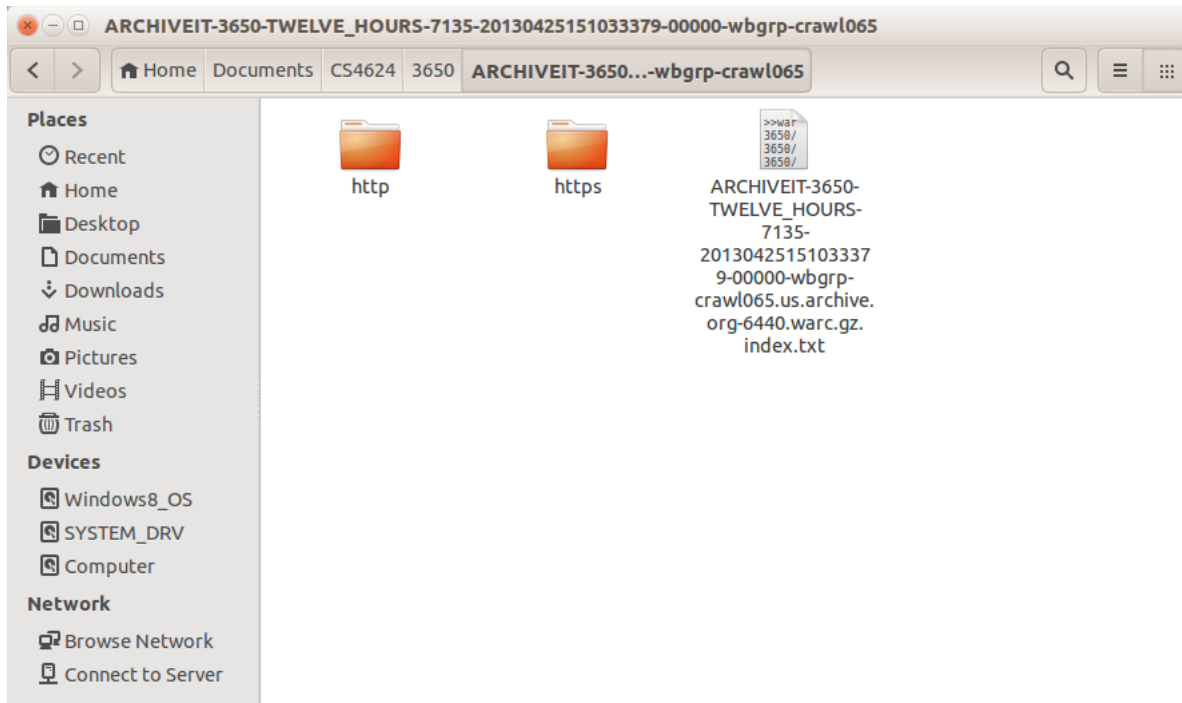


Figure 3: Directory after Web Archive files have been processed

Individual Functions in Script

This section is for anyone interested in making use of the individual functions that are used in this script.

The following provides a brief overview of the functions and their applications. To examine the code,

one can view the `processWarcDir.py` script that is included with this document. Note that these

functions are also described in the Developers Manual as well.

parseLogFileForHtml(log_file):

This function takes text file as its input and returns a list that holds the location of every HTML file found from this text file. The text file used for this function results from running `processWarcFile()`.

It is important to note that if the user desires to unzip the web archive files separately from the rest of the process, using this function independently provides a useful way to do so.

processWarcFile(warc_file):

This function primarily makes the use of hanzo-warc-tools script. If the user desires to only unpack a web archive file for all of its contents, the use of this function with hanzo-warc-tools provides a useful way to do so.

extractTextAndIndexToSolr(html_file, file_url):

This function takes an HTML file and removes the tags and indexes the raw text into a Solr instance. If a user wishes to use this they should ensure they are doing so concurrently with the use of BeautifulSoup. This function could be used in isolation if a user wished solely to extract text from HTML files and index them into Solr.

Notes

- When installing Python, ensure that there is only one installation of Python and that it is located in `usr/bin/python`.
- Running the script with Hadoop requires modifications to the command line procedures.
- BeautifulSoup and Hanzo Warc tools are not considered commercial software and users who download these tools should be aware of that.
- The independent functions may not run due to the fact that some of the functions make use of each other in order to operate.

Developer's Manual

The purpose of this developer's manual and guide is so that another developer can continue any work on this project in progress, and expand on it as necessary. In order to achieve this goal, the Developer's Manual contains the following:

- Figures describing:
 - Project Architecture
 - Project Flow
- Descriptions of Program Files:
 - The concurrent script - processWarcDir.py
 - The Hadoop scripts - mapHTML.py and mapSolr.py
- Project Dependencies
- Concept Map covering Hadoop
- Possibilities for expansion

Project Architecture

Overall IDEAL Project Architecture (1)

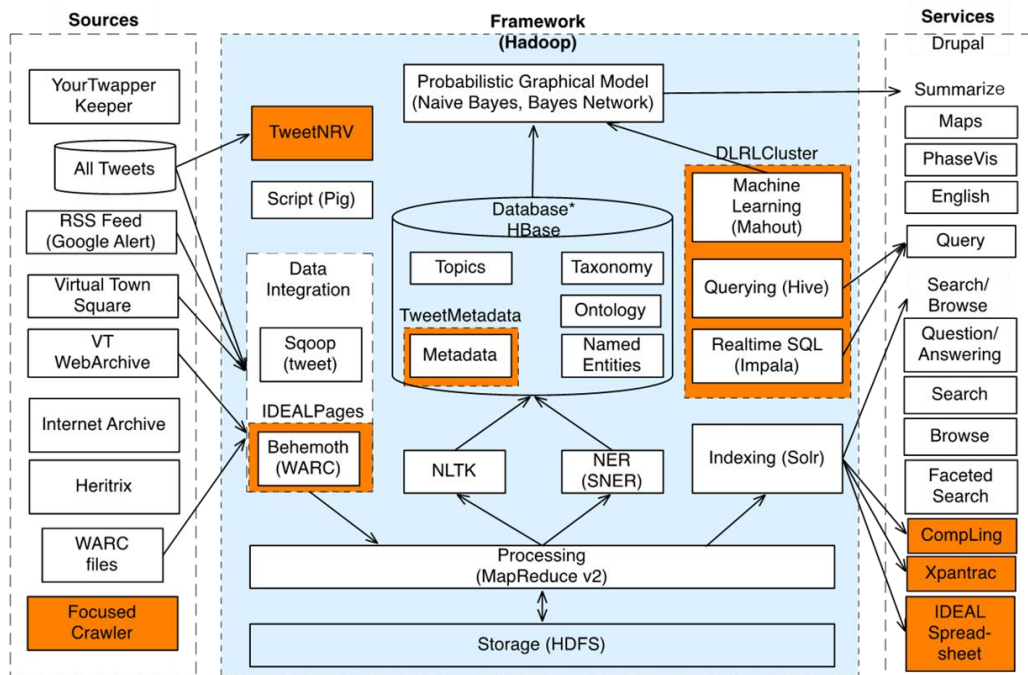


Figure 4: IDEAL Prproject Architecture

The above table gives an overall description of the IDEAL Project along with several other projects. IDEAL Pages is shown on left, and its impact on the other projects can be seen by following the arrows across the diagram.

Project Flow

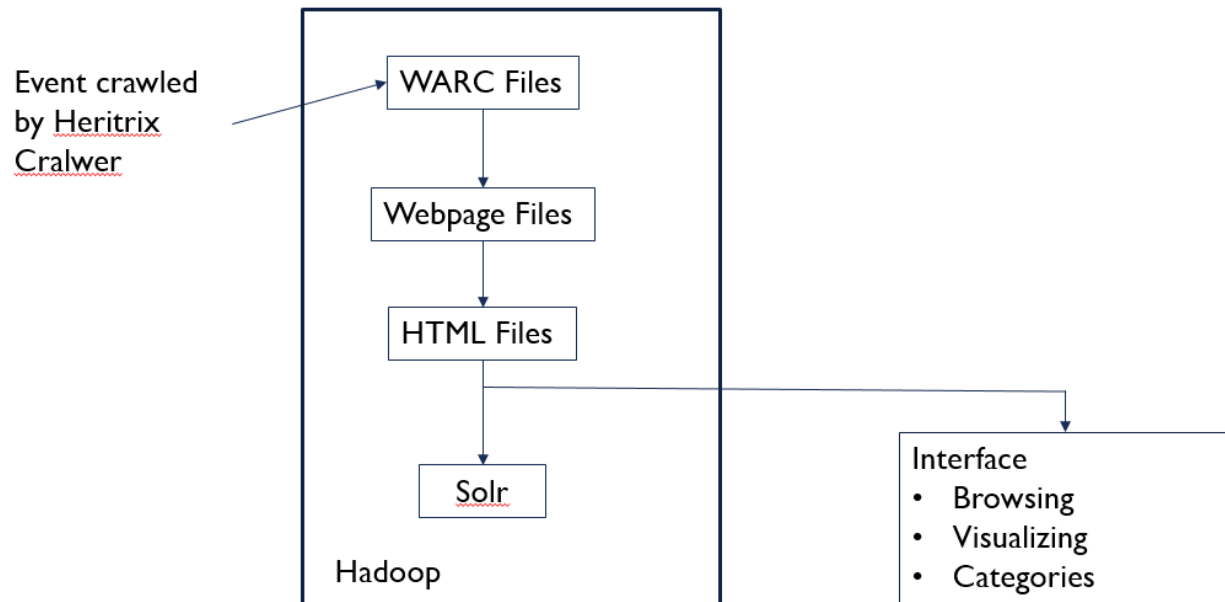


Figure 5: IDEAL Pages project flow

Figure 5 describes in greater detail the flow that occurs during this project. As can be seen in the figure, a web crawler produces the web archives needed, which are then extracted into their webpage files, followed by the filtering HTML files that are indexed into Solr, and a web interface (completed by a different project) for viewing the indexed information.

Program Files

This project is made up of several Python files. There is one main file for executing the process concurrently and undistributed on Hadoop. Python scripts for distributing across Hadoop consist of the functions provided in the concurrent script rewritten for use on Hadoop.

Concurrent Script

The User's Manual previously described how to run the concurrent script, `processWarcDir.py`, but just to reiterate, this script focuses on processing a directory or directory of directories of warc files. Several functions make up this script. The main function walks along the provided directory searching for warc files, on which it calls `processWarcFile()` on. This function handles a few parts of setup for unpacking the provided warc file, and then calls `unpackWarcAndRetrieveHtml()`. This function does exactly what its name says. It unpacks the given warc file into `http` and `https` folders and creates an index text file listing all content extracted. A function, `parseLogFileForHtml()`, then takes in this index file and returns a list of the HTML files extracted from the warc file and other information necessary for indexing into Solr. Next, `extractTextAndIndexIntoSolr()` is called for every html file found. This function extracts text for indexing into Solr and then uses the declared Solr instance for indexing valid Solr documents.

Hadoop Scripts

The two provided Python scripts, `mapHTML.py` and `mapSolr.py`, are mappers for the MapReduce framework used on Hadoop. Both scripts read from standard input, so they rely on proper command prompt execution in order to give the proper files. `mapHTML` expects standard input to contain the information written in an index file created from the extraction of a warc file. The output produced from `mapHTML` are the locations of HTML files that were previously extracted. Since there is nothing to reduce at this step, the output is used by `mapSolr` now. This script reads every line of standard in, opens the file specified by that line, extracts the text and indexes the information into Solr. The framework used for interacting with Hadoop is explained further in the next section.

Hadoop Concepts (MapReduce)

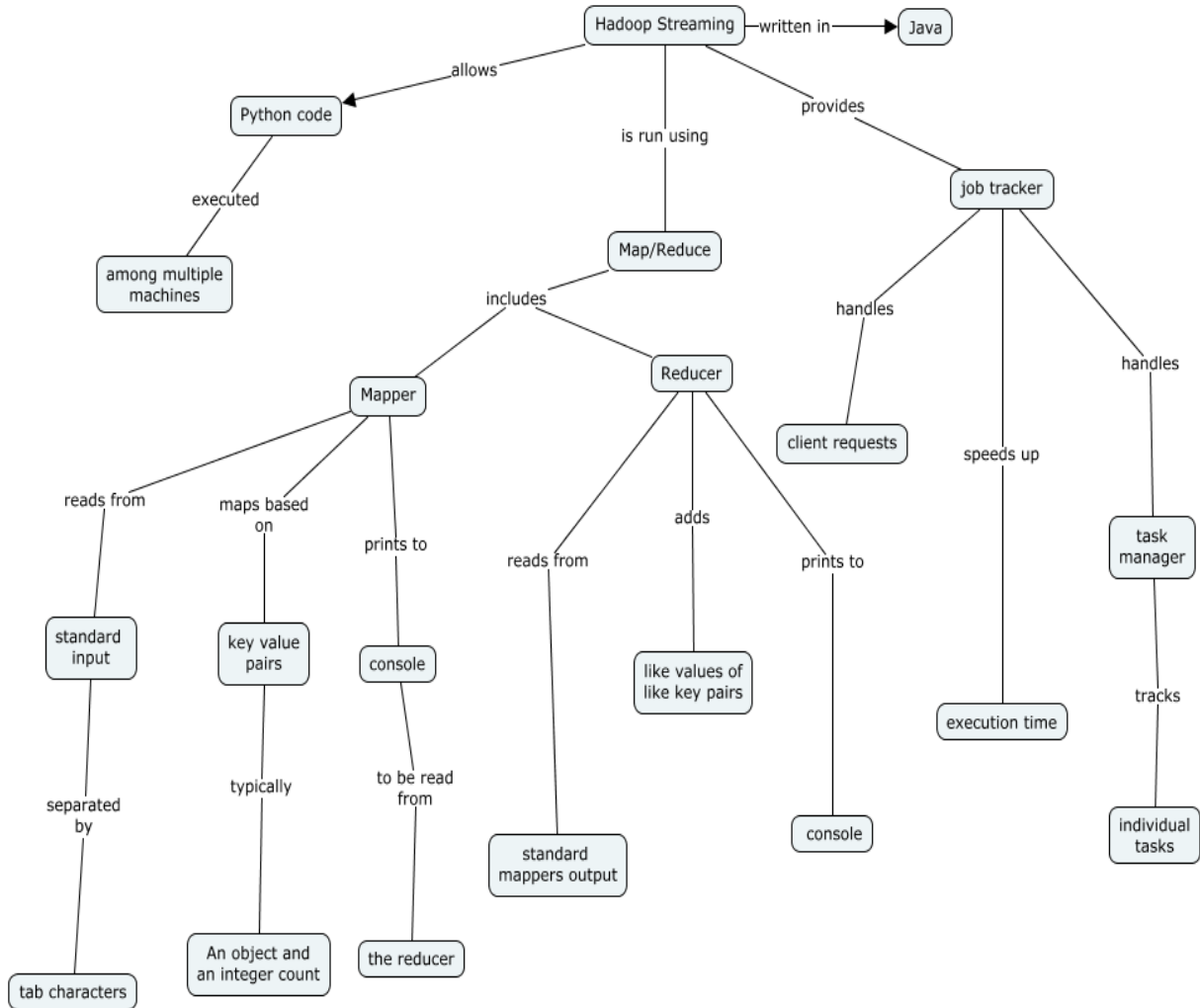


Figure 6: Hadoop Concepts

The above concept map gives an illustration of how Hadoop Streaming operates. It is key to note that both the mapper and the reducer read from standard input. In relation to IDEAL Pages, this was a hurdle to overcome since none of the tasks involved reading from the console. However by piping the text file to the console and reading line by line, we were able to abide by the standards that Hadoop requires.

Project Dependencies

Solrpy, BeautifulSoup4, and Hanzo Warctools make up the three external dependencies used in this project. Solrpy was chosen because it provided an easy-to-use API for interacting with Solr. There are three functions utilized from this API including `solr.SolrConnection()` for initiating the connection to a Solr server, `add()` for adding a document to the index, and `commit()` for committing the additions to the Solr server (2). Next, BeautifulSoup4 makes text extraction from html files much simpler. It provides ways of extracting important information from the files using the structure of the html (3). Finally, Hanzo Warctools gives the project the capability of unpacking web archives and creating an index of all the files found in the archive (4). This functionality is pertinent to the capabilities of this project. Any developer continuing work on this project should familiarize themselves with these tools in order to continue work. It is also suggested that these tools be kept up to date in order to continue to get the most value out of this project.

Possible Expansions

The project presented has not yet reached the full potential. There are several extensions for this project that a developer or client would find useful. Some possibilities for expanding upon this project include distributing the process further across a Hadoop cluster, generalizing the code base for reuse on similar tasks, and exploring the possibilities for other files extracted from web archives.

Currently, the only processes that have been distributed across Hadoop include the filtering of html files from the expansion of web archives and the indexing of text into Solr. The first task of expanding and extracting the web archives has yet to be distributed across Hadoop through the MapReduce

framework. This presents a quality and valuable opportunity to expand on this project. By distributing this part of the process, there would be an expected speed increase since it is currently the most involved step of the process.

The project described in this report focuses on completing a task for the purpose of indexing webpages for the IDEAL project. This purpose has led to various parts of the code that focus mainly on achieving this goal rather than providing a general solution. Some ways this project could be generalized include:

- Customization of Solr index fields – certain fields exist for the purpose of the IDEAL project
- Easier access to modifying the URL used for Solr

Of course these are not the only ways of generalizing this project, but they make it more possible to reuse the project for similar problems.

Web archives contain all files necessary for a webpage to function properly, not just the html files. This includes images, style sheets, and other resources. These resources present an opportunity to make use of the majority of the code base already in existence. The process of filtering html files has been setup in such a way that the file's searched for only need to be changed to retrieve more than just html files.

Overall, there are several opportunities to expand and reuse this project beneficially.

Lessons Learned

This project provided a challenging opportunity to explore new technologies and tools that we had never worked with before. The project took the entirety of the Spring 2014 semester, and will likely be expanded upon as the IDEAL Project progresses. With the IDEAL Pages project being such a critical piece to the puzzle that is the IDEAL Project, the team strived to find solutions that would make integration with the other pieces smooth. But with a project of such scale, it is no surprise that issues may arise

which halt the progress and hinder the development process. This section aims to go over these issues and the team's strategy to discover solutions.

Timeline

Initial Schedule:

Date	Milestone to Complete
February 15 th	Ensure all tools for development are installed. Acquire a sample .warc file for examination. Become familiarized with SOLR and development environment.
March 1 th	In Python, be able to extract from a .warc file and read contents.
March 15 th	Integrate a parser to filter out all non html files. Be able to extract text from the html files.
April 1 st	Integrate software to be run with all .warc files. Index files into SOLR.
April 15 th	Connect back end software with Hadoop and the front end interface (Created by 6604 students)
May 1 st	Conduct regression testing and make sure all functionality is working as expected.
Final	Seek client out and ensure project was complete to satisfaction.

Figure 7: Initial project schedule

Reflection on Initial Schedule:

Looking back at the original schedule set for this project the tasks were spread out fairly well throughout the semester. In retrospect, leaving more wiggle room at the end of the semester for testing would have greatly helped the project roll smoothly into the end of the semester. Too much time was allocated for extracting a web archive file and extracting the HTML from it. These steps were fairly simple to put together and the majority of the time spent on these tasks was researching the best tools to use. Unseen circumstances caused some areas to take longer than others, below is the way the schedule actually panned out using the same dates as benchmarks.

Final Schedule:

Date	Milestone to Complete
February 15 th	Install Python environment, learn process in place of unzipping contents of a web archive file.
March 1 th	Research tools to extract HTML files from an unpacked archive file.
March 15 th	Integrate a parser to filter out all non html files. Be able to extract text from the html files.
April 1 st	Integrate software to be run with all .warc files. Index files into Solr.
April 15 th	Begin research on Hadoop and Map/Reduce, clean up Solr process
May 1 st	Begin developing scripts for Map/Reduce using Hadoop and Cloudera
Final	Finish building Map/Reduce scripts. Seek client out and ensure project was complete to satisfaction.

Figure 8: Final Project Schedule

Reflection on Final Schedule:

The project had a rough start, ran very smoothly midway into the semester, and became difficult again near the end of the semester. Unfortunately for this project's purposes, simply installing Python was not enough to begin serious development. The Hanzo Warc tools that we wanted to incorporate was not cooperating with the installing of Python on the machines used to develop. Working with Mohamed Magdy to get these tools installed, the project was able to continue. A fair amount of research went into tools to extract HTML files and remove the tags. Eventually a program called Beautiful Soup was found and again, problems were faced integrating that into the environment being used. The biggest issue with the final schedule came when we tried to incorporate Hadoop to distribute the process across the cluster. It was later found that what makes Hadoop such a valuable tool is its use of Map/Reduce. Map/Reduce requires a mapper and reducer for each step in the process. This requires a full makeover of the original script and given the tasks in this project, was not intuitive. Writing the scripts with a mapper and reducer from the start would have proved to be a better solution than what was done. Overall however, the project ran smoothly and a lot of was learned in the process. The final schedule did not deviate too much from the original schedule that was planned.

Problems/Solutions

As with any software development project, many issues arise that are unexpected, some taking much longer to remedy than imagined. With the use of so many new technologies, pieces of software, and tools, the whole experience becomes a learning process. Originally, the plan was to develop in Java, but as more research on the task was done, it became clear that Python was the best option. Drastic changes such as this cause a delay in production and a frustrating experience for all involved. What mattered more is that when these problems arose, the team sought assistance when needed and found solutions to the problems. Detailed below are the problems that were faced, with the next section focusing on the solution to those problems.

Trouble setting up development environment:

Problem: After some time, the team concluded that Python would be the best language to develop in. This is due mainly to the fact that many of the tasks that needed to be done could be written in a fraction of the lines that it would take in another language such as Java. However, with this great positive feature, came the struggle of integrating the environment. After installing Python on the team's local machines, one of the outside scripts to be used would not run properly. This problem took something that was anticipated to be a two hour task and turned it into a week-long task.

Solution: After working with the Python environment for a little bit and trying to run some simple scripts, the team faced problems with the development environment. After searching online for a solution to no avail, we decided to seek assistance from our client, who is one of the leading efforts in the overall IDEAL Project. After working with him for an hour, we were able to figure out the problem and get the scripts we wanted running on our local machines.

Working in a new programming language:

Problem: No one on the team had any experience working with Python. Python is an easy language to pick up, but when the time frame is only a few months, it can still be difficult to learn. Studying the libraries that were available in this language took some time, and at some points, made making progress on the project a struggle.

Solution: To solve this problem, we researched some of the different libraries available and learned how to do tasks we already knew in other languages to see how they compared. Shortly after, we began slowly writing code and testing regularly to make sure we weren't making simple mistakes. Using this incremental development process, we were able to easily overcome the hurdle of working in a new language.

Local Machine Failure:

Problem: Machine failure is not something that can be anticipated, but it is something that happens nonetheless. A minor problem in the projects development was the main local machine that the group was working on had a full breakdown. This caused a loss in the development environment that took so long to construct. No code was lost in this, but it was a bump in the road.

Solution: Midway through the semester the main computer which work was being done on completely failed. This included losing the computer as a functional tool as well as the hard drive. While this was a small issue, it did force us to look back at the process it took to originally set up the development

environment. Luckily the second time around was much easier and this issue was quickly solved because we recorded the steps we used to originally set up the work environment.

Dedicating time:

Problem: As seniors in the Computer Science program, many other classes require attention. Finding times to meet and make progress on the project were sometimes difficult. Great discipline was required to ensure that deadlines were met and progress continued.

Solution: To stop other classes getting in the way of us making progress on the project, there were two main courses of action that were taken. The first was to set a weekly meeting time with our client, Mohamed Magdy. After we had added that into our schedule, we were able to realign our focus every week and know exactly what was expected out of us and what to accomplish on a week to week basis. This proved to be extremely helpful near the end of the semester when other classes were becoming extremely time consuming as well. The second course of action was to set hard deadlines for certain tasks. This forced us to see completing parts of the project as priorities in our schedule, rather than waiting until the last minute to complete tasks. We found that these two things meshed together well as Mohamed would consistently ask us for time frames on different parts of the project.

Issues integrating Map/Reduce:

Problem: Possibly the toughest task and biggest problem in the project was integrating Hadoop and Map/Reduce into the project. As stated in the schedule portion of the report, the best plan would have been to keep in mind the capabilities and needs of Map/Reduce from the start of the project.

Unfortunately, trying to integrate this at the end was a struggle and there seemed to be limited resources that related to the task that we had. Finding a way to make both a mapper and reducer for each individual task was a very unique issue. For Map/Reduce, the scripts typically take input from standard in and print to the console. The reducer generally sums up some value from what is read from standard in and reduces the process in that regard. For the reduction phase in our project, we had trouble finding a logical need for the reducer. Once that issue was solved, we struggled to find the correct way to run the scripts and get proper output from the server Hadoop was installed on.

Solution: To solve the issue of integrating Map/Reduce with our initial script proved to be quite difficult. The first step for us was to research as much as possible on Map/Reduce. We were able to find many examples and articles about the tool but still struggled to conceptualize it in relation to our project. After a couple more meetings with Mohamed we finally were able to conclude that it would not be necessary for there to be a reduce step in the process. We came up with two mapper scripts, one to extract the HTML files from a web archive file, and a second to extract text and index those files into Solr. The final step in solving this issue was to find a way to test and run the scripts on Heritrix (the machine at the head of the cluster). Looking through more tutorials we were able to find a set of commands that would run the scripts on Hadoop Streaming. While there is still more testing and integration left, we had made great strides in the final step to integrating the full project onto the cluster.

Acknowledgements

The authors would like to thank their client Mohamed Magdy for his support and guidance during the undertaking of this project. Mohamed Magdy is currently a PhD student at Virginia Tech and part of the

Digital Libraries Research Laboratory with Dr. Edward Fox. Mohamed may be contacted at his email: mmagdy@vt.edu. In addition, we would like to acknowledge Dr. Edward Fox and everyone involved in the efforts of NSF IIS -1319578: Integrated Digital Event Archiving and Library (IDEAL).

References

1. **Lee, Sunshin.** *Big Data and Hadoop*. [PowerPoint] Blacksburg : Virginia Tech, 2014.
2. **Python Solr Module.** *Google code*. [Online] <https://code.google.com/p/solrpy/>.
3. **Richardson, Leonard.** Beautiful Soup. *Crummy Software*. [Online] 10 18, 2013. <http://www.crummy.com/software/BeautifulSoup/>.
4. **Hanzo.** Hanzo Archives. *Bitbucket*. [Online] <http://code.hanzoarchives.com/warc-tools>.