DATASCI W205: Storing and Retrieving Data – Final Paper

Document Overload, Finding a Needle in a Haystack

Jason Blauvelt, Geoffrey Link, Megan Pera, Guangzhi ‘Frank’ Xie

**Why this is an Exciting Project![[1]](#footnote-1)**

The cost of discovery in a legal investigation is significant; for example, when five television studios became entangled in a Justice Department lawsuit, the studios examined 6 million documents at a cost of more than $2 million. In contrast, Blackstone Discovery of Palo Alto, California helped analyze 1.5 million documents electronically for less than $100,000.

The key in this type of analysis is not only to recognize relevant words, but also relevant concepts; for example, a shift in an email author’s style from breezy to unusually formal can raise a red flag about illegal activity. The most obvious being the acronym “LTOL”, also known as, Let’s Take This Offline.[[2]](#footnote-2)

Discovery technologies generally fall into two broad categories: linguistic and sociological. We focus primarily on the most basic linguistic approach to analysis: using specific words to find and sort relevant documents. Sociological approaches add a layer of deduction and attempts to determine the interactions of people to find anomalies in behavior; in this manner, we attempt to build a basic network analysis using Neo4J.

In brief, the productivity gains represented by our project are unimaginable.

**Motivation**

The key motivation for this project is the financial crisis of 2008 in which regulators did not have enough information to understand the breadth and depth of the crisis and its effect on financial institutions and the economy as a whole.

Data obfuscation is a key tactic of organizations under investigation or regulation. For example, Goldman Sachs Group Inc. was subpoenaed by the Financial Crisis Inquiry Commission (FCIC) after the FCIC accused the firm of engaging in a document dump to hinder a probe. Goldman Sachs sent more than a billion pages of documents, not all of which is the information that the FCIC requested. In essence, transmitting such a large amount of information in a time constrained investigation is a deliberate attempt to run out the clock such that no tangible insights can be obtained from the information before the investigation ends[[3]](#footnote-3).

The analysis of large volumes of data is a near impossibility for organizations that face both time and budget constraints. Hiring hundreds of $200 per hour junior lawyers to shuffle through a billion pages of documents is an option only a few organizations can afford; moreover, this process is subject to human error and fatigue.[[4]](#footnote-4)

To sum up, we propose to create a proof-of-concept implementation of a solution to this issue utilizing the tools we learned in W205.

**Data Sources and Acquisition**

For the purposes of this project, we chose to use the publicly available Enron Email Data from the Federal Energy Regulatory Commission (FERC) available on Amazon Web Services.[[5]](#footnote-5) Enron’s complex financial statements were confusing to shareholders and analysts; off-balance-sheet vehicles, complex financing structures and deals were so confusing that few people could understand them.[[6]](#footnote-6) Enron’s auditor, Arthur Andersen, was found guilty of illegally destroying documents relevant to the SEC investigation. Similar to financial institutions after the financial crisis of 2008, Enron elected to disclose minimal details on its use of special purpose entities; example of which include: JEDI, Chewco, Whitewing, and LJM.

The Enron Email Data is unique is that it is one of the only publicly available collections of real emails easily available for study as such collections are typically bound by numerous privacy and legal restrictions which render them difficult to access.[[7]](#footnote-7)

The volume of data in the Enron dataset is significant at a total of 210Gb representing emails transmitted among 151 senior staff at Enron between the years 1998 and 2002.[[8]](#footnote-8)

The variety of data types in the Enron dataset is significant as we encountered a number of different and proprietary file types: .csv, .bmp, .bz2, .doc, .eml, .gif, .pdf, .jpg, .ppt, .pst, .tif, .txt, .url, .wpd, .xls, .xml, .z0x, .zip

For the purposes of this project and in the interest of time and money, we focused solely on the text-based and xml-based data formats.

To collect longitude and latitude information we utilized GeoPy. As the GeoPY API presented us with our only rate-limiting challenged, we used a combination of Kafka and Storm to slow retries to account for any data collection restrictions encountered.

**Data Storage, Processing, and Presentation**

When we began this project, we did not have a strong grasp of the tools available to us. As the project and course progressed, we learned a great deal about working with large datasets, especially large datasets of millions of small files, which does not appear to fit well within the Big Data paradigm.

* **Winzip**

The Enron Email Data came in zipped format, which was not ideal for processing as we encountered a number of winzip errors as well as uuencode errors. Thus, we needed to experiment with the zipped data in order to get it into a format that could be processed easily and in a timely fashion.

* **Linux**

The Linux operating system began to break down once we began processing millions of small files. We often received the error ‘argument list too long’ with basic Linux commands such as ‘ls’ and ‘rm’, as well as exceptionally slow response times from these commands. To resolve these issues, we used a bucketing approach to the files based upon the Enron staffer’s last name, first initial. This generally separated the emails into directories of 100,000 email files or less.

* **Python**

Python is the workhorse of our project as it proved lightweight and efficient at manipulating large numbers of files. We believe the only downside to Python is an inability to easily parallelize operations outside of a 3rd party technology such as Storm or Spark.

* **Kafka**

Kafka proved itself to be an efficient and lightweight messaging bus.

* **Storm**

We used Storm to slow the processing of requests via GeoPy as we often received timeout errors as the live source would cut our connection after too many calls. Thus, when dealing with a data source that is much slower than other sources, we felt it best to either download all of the data from the source in a single pull (if possible), or slow the requests by dumping the requests into a Kafka queue and then allowing Storm to make requests as time permits and then upsert the requisite database tables.

* **Spark**

Spark proved exceptionally helpful at parallelizing many of our operations; however, Spark was often subject to random errors that were not well documented as well as Memory errors that seemed to be only resolvable via a larger node size.

* **MongoDB**

MongoDB was an attempt at storing and processing a significant amount of email data; however, we found that MongoDB was highly inefficient as it consumed significant amounts of CPU and Memory. Our initial analysis of a small number of records with MongoDB proved sufficient; however, performing iterative analysis on millions of records proved highly inefficient. Using a m3.medium instance required 2 hours and 10 minutes of processing time to loop through each xml document. We submit that running a sharded MongoDB on Amazon EMR could provide a significant improvement to this process time.

* **Hadoop**

Hadoop does not perform well with millions of small files as by default Hadoop allocates a minimum of 64Mb per file and creates 3 copies of each block of data.

* **Postgres**

We solely used Postgres as a data summarization tool as we did not feel that Postgres was the appropriate tool to use for large data manipulation.

* **Neo4j**

**Tools and Third Party Libraries**

* psycopg2
* AWS CLI (Command Line Interface)
* pymongo
* xml.etree.ElementTree
* xmltodict
* textblob
* geopy
* kafka
* streamparse

**Architectural Consideration**

**Amazon EC2/EBS Configuration**

For lightweight data processing, persistent database storage, and data presentation we used a m3.medium instance as our core processing occurred in the Amazon EMR configuration described below.

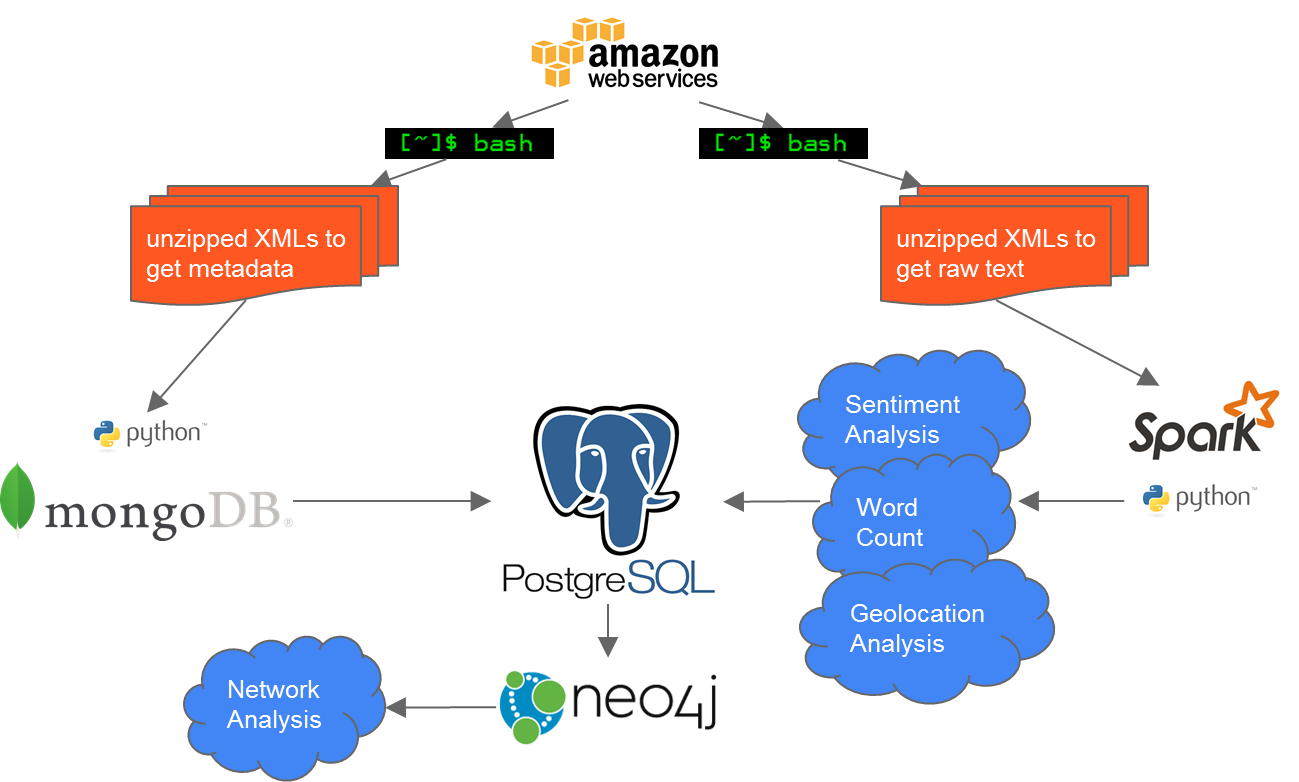
**Amazon EMR Configuration**

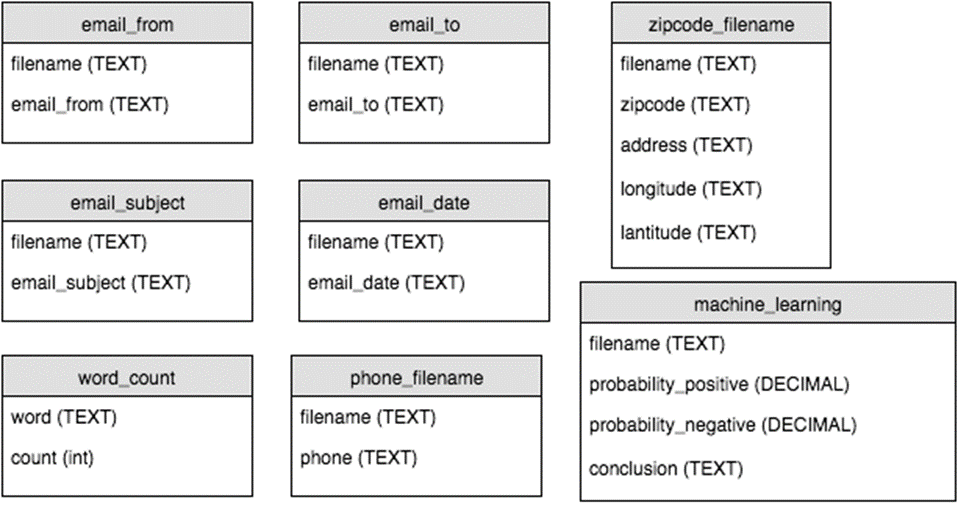
For heavier, transient data processing we utilized a cluster of m3.xlarge instances consisting of 3 worker nodes and 1 master node.

**Amazon S3**

We choose to use S3 to easily transfer data from our persistent EC2 instance to our transient EMR cluster and vice-versa. Although S3 does not perform as well as Hadoop, we found the flexibility of S3 much more attractive than EBS.

**Architecture, Process Flow, and Implementation Strategy**





**Analysis Results**

**Visualization and User Interface**

**Lessons Learned**

* sc.binaryFiles
* Storm - pitfalls of open source
* Tool selection is essential

**Future Roadmap**

**Appendix**

<https://github.com/jablauvelt/DocumentOverload_205_Project>

1. New York Times, Armies of Expensive Lawyers Replace by Cheaper Software, nytimes.com/2011/03/05/science/05legal.html [↑](#footnote-ref-1)
2. MIT Technology Review, The Immortal Life of the Enron Emails, technologyreview.com/s/515801/the-immortal-life-of-the-enron-e-mails/ [↑](#footnote-ref-2)
3. Bloomberg, Goldman Subpoenaed After FCIC Says Firm Slowed Probe, 2010. bloomberg.com/news/articles/2010-06-07/goldman-sachs-documents-subpoenaed-by-u-s-financial-crisis-investigators [↑](#footnote-ref-3)
4. CNN, The fall and rise of lawyers, 2015. cnn.com/2015/05/22/opinions/barton-rise-and-fall-of-lawyers/ [↑](#footnote-ref-4)
5. Amazon Web Services, Enron Email Data, aws.amazon.com/datasets/enron-email-data/ [↑](#footnote-ref-5)
6. Wikipedia, Enron scandal, en.wikipedia.org/wiki/Enron\_scandal [↑](#footnote-ref-6)
7. Wikipedia, Enron Corpus, en.wikipedia.org/wiki/Enron\_Corpus [↑](#footnote-ref-7)
8. Journal of Statistics Education, Network Analysis with the Enron Email Corpus, amstat.org/publications/jse/v23n2/hardin.pdf [↑](#footnote-ref-8)