

# CS 5604: Information and Storage Retrieval

Term Project: Collection Management Tweets

Final Report

Faiz Abidi <sup>\*</sup>, Shuangfei Fan <sup>†</sup>, and Mitchell Wagner <sup>‡</sup>

Instructor: Dr. Edward A. Fox

*Department of Computer Science, Virginia Tech, Blacksburg VA 24061*

Dated: December 19, 2016

---

<sup>\*</sup>Electronic address: `fabidi89@vt.edu`; Corresponding author

<sup>†</sup>Electronic address: `sophia23@vt.edu`; Corresponding author

<sup>‡</sup>Electronic address: `mitchw94@vt.edu`; Corresponding author

## Abstract

The goal over the 2016 Fall semester was to address the following problem statement:

**“How can we best build a state-of-the-art information retrieval and analysis system in support of the IDEAL (Integrated Digital Event Archiving and Library) [1] and GETAR (Global Event and Trend Archive Research) [2] projects.”**

The entire project was divided into six parts - Classification (CLA), Collection Management Tweets (CMT), Collection Management Webpages (CMW), Clustering and Topic Analysis (CTA), Front End (FE), and Solr (SOLR). This report documents the work done by the CMT team.

To give a high-level view, we were responsible for processing 1.2+ billion tweets, including data transfer, noise reduction, tweet augmentation, and storage via several technologies. The processed tweets were to be used by other teams like the front end and the Solr teams to further build the project. We were also responsible for building a social network (or set of networks) for those tweets, along with their tweeters.

We utilized and built off of the work done by students in the previous semesters of this class. Some of our project tasks included:

- Adding an incremental update feature in the tweets extraction process - Previously, this update only happened in batches. This effectively entailed a wholesale, bulk update of tweets from the MySQL database to HDFS [3]. We made this a dynamic process in which new tweets get updated in the database incrementally every day
- Adding incremental loading from HDFS to HBase [4] - HDFS stores the cleaned tweets, but we needed an efficient way to also load the new tweets coming in daily into the HBase database.
- Removing noisy, irrelevant, or unwanted data (profanity words, irrelevant tweets, broken links, etc.) - Although the existing system handled this problem to a certain extent, our goal was to optimize those solutions for sequential updates, and further refine the tweets with additional filtering if time permits.

We needed to research and understand the existing software pipeline before starting any development work, including the technologies that we would be using to move and clean the tweets. We also had to research other open source tools that could have been a better choice to do this work.

In addition to the aforementioned tasks, our responsibilities also included building a social networks of tweets. This entailed doing research into that space and determining what type of graphs would be most appropriate. We also needed to research methods ascribing importance to nodes and edges in our social networks once they were constructed, and analyze our networks using these techniques. Finally, we had to closely collaborate with the other teams, especially the front end and the Solr teams, who depended on our input to build their solution.

We worked with a small subset of data initially that served as a proof-of-concept that our solution works. After that, we tested our solution with a bigger data set to see if our solution scaled properly, and it did for a collection of 155000 tweets.

# Contents

|   |           |
|---|-----------|
| <b>List of Tables</b>   | <b>6</b>  |
| <b>List of Figures</b>  | <b>7</b>  |
| <b>1 Requirements</b>   | <b>9</b>  |
| 1.1 Overview of the IDEAL and GETAR projects . . . . .            | 9         |
| 1.2 Current system setup . . . . .                                | 11        |
| 1.3 Functionality . . . . .                                       | 12        |
| 1.3.1 Incremental database updates . . . . .                      | 12        |
| 1.3.2 Tweet cleaning, noise reduction, and augmentation . . . . . | 13        |
| 1.3.3 Building a tweet and webpage social network . . . . .       | 14        |
| 1.4 Input and output . . . . .                                    | 14        |
| 1.4.1 Input . . . . .   | 14        |
| 1.4.2 Output . . . . .  | 14        |
| 1.4.3 Collaborations . . . . .                                    | 14        |
| 1.5 Level of performance . . . . .                                | 15        |
| 1.6 User support . . . . .  | 15        |
| <b>2 Overview of project effort</b>                               | <b>15</b> |
| 2.1 Project management . . . . .                                  | 15        |
| 2.1.1 Weekly meetings . . . . .                                   | 15        |
| 2.1.2 Communication . . . . .                                     | 15        |
| 2.1.3 File-sharing . . . . .                                      | 15        |
| 2.2 Problems and challenges faced . . . . .                       | 15        |
| 2.2.1 Constraints . . . . .                                       | 15        |
| 2.3 Additional challenges . . . . .                               | 16        |
| 2.4 Solutions developed . . . . .                                 | 16        |
| 2.4.1 First report . . . . .                                      | 16        |
| 2.4.2 Second report . . . . .                                     | 17        |
| 2.4.3 Third report . . . . .                                      | 17        |
| 2.4.4 Final report . . . . .                                      | 18        |
| 2.5 Future work . . . . .   | 19        |
| <b>3 Literature review</b>  | <b>21</b> |
| 3.1 Database updates and backups . . . . .                        | 21        |
| 3.1.1 Cold database backups . . . . .                             | 22        |
| 3.1.2 Hot database backups . . . . .                              | 23        |
| 3.1.3 Full backup . . . . .                                       | 23        |
| 3.1.4 Incremental backup . . . . .                                | 24        |
| 3.1.5 Differential backup . . . . .                               | 25        |
| 3.2 Incremental update from relational database to HDFS . . . . . | 26        |
| 3.2.1 Percona Toolkit . . . . .                                   | 26        |
| 3.3 Incremental update from HDFS to HBase . . . . .               | 27        |
| 3.4 Text cleaning and noise reduction . . . . .                   | 27        |

|          |   |           |
|----------|---|-----------|
| 3.4.1    | Text cleaning . . . . .   | 27        |
| 3.4.2    | Noise reduction . . . . .   | 27        |
| 3.5      | Social networks . . . . .   | 30        |
| 3.5.1    | The PageRank algorithm . . . . .  | 33        |
| <b>4</b> | <b>Design</b>   | <b>35</b> |
| 4.1      | Approach . . . . .  | 35        |
| 4.2      | Tools . . . . .   | 35        |
| 4.3      | Methodology . . . . .   | 35        |
| 4.4      | Conceptual background . . . . .   | 35        |
| <b>5</b> | <b>Implementation</b>   | <b>36</b> |
| 5.1      | Timeline . . . . .  | 36        |
| 5.2      | Details on the test data used for this project . . . . .                        | 37        |
| 5.3      | Deriving our solution . . . . .   | 38        |
| 5.3.1    | Incremental database updates . . . . .  | 38        |
| 5.3.2    | Noise reduction and cleaning . . . . .  | 45        |
| 5.3.3    | Social network . . . . .  | 53        |
| <b>6</b> | <b>User manual</b>  | <b>61</b> |
| 6.1      | Incremental Update from MySQL to HDFS . . . . .                                 | 61        |
| 6.1.1    | Transfer SQL data to the MySQL database . . . . .                               | 61        |
| 6.1.2    | Use pt-archiver to transfer tweets to the ArchiveDB, and save to a file . . . . | 62        |
| 6.1.3    | Cleaning the text file of tweets . . . . .                                      | 64        |
| 6.1.4    | Converting the CSV file into an Avro file and copying it to HDFS . . . . .      | 65        |
| 6.1.5    | Merging the Avro files on HDFS . . . . .  | 66        |
| 6.2      | Incremental update from HDFS to HBase . . . . .                                 | 67        |
| 6.2.1    | Preparing the pipeline . . . . .  | 67        |
| 6.2.2    | Running our scripts . . . . .   | 67        |
| 6.3      | Build social network . . . . .  | 68        |
| 6.3.1    | Collect data . . . . .  | 68        |
| 6.3.2    | Use Twitter API . . . . .   | 68        |
| 6.3.3    | Build social network and compute importance factor . . . . .                    | 70        |
| 6.3.4    | Visualization using NetworkX . . . . .  | 71        |
| <b>7</b> | <b>Developer manual</b>   | <b>73</b> |
| 7.1      | IDEAL/GETAR cluster architecture . . . . .                                      | 73        |
| 7.2      | Module overviews . . . . .  | 73        |
| 7.2.1    | MySQL database . . . . .  | 73        |
| 7.2.2    | MySQL installation & operation . . . . .  | 76        |
| 7.2.3    | Apache Hadoop . . . . .   | 77        |
| 7.2.4    | Apache Hadoop installation & operation . . . . .                                | 78        |
| 7.2.5    | HDFS . . . . .  | 80        |
| 7.2.6    | HDFS installation & operation . . . . .   | 81        |
| 7.2.7    | Apache HBase . . . . .  | 82        |
| 7.2.8    | Apache HBase installation & operation . . . . .                                 | 83        |



|          |   |            |
|----------|---|------------|
| 7.2.9    | Apache Sqoop . . . . .                          | 85         |
| 7.2.10   | Apache Sqoop installation & operation . . . . . | 85         |
| 7.2.11   | Apache Pig . . . . .                            | 87         |
| 7.2.12   | Apache Pig installation & operation . . . . .   | 88         |
| 7.2.13   | Pt-archiver installation & operation . . . . .  | 89         |
| 7.2.14   | csv2avro . . . . .                              | 90         |
| 7.2.15   | csv2avro Installation & Operation . . . . .     | 90         |
| 7.3      | Project installation . . . . .                  | 92         |
| 7.4      | Software versions . . . . .                     | 96         |
| 7.4.1    | MySQL to HDFS incremental update . . . . .      | 96         |
| 7.4.2    | HDFS to HBase incremental update . . . . .      | 97         |
| 7.4.3    | Social network . . . . .                        | 97         |
| 7.5      | File inventory . . . . .                        | 98         |
| <b>8</b> | <b>Acknowledgments</b>                          | <b>99</b>  |
| <b>9</b> | <b>Appendix</b>                                 | <b>100</b> |
| 9.1      | Table of acronyms . . . . .                     | 100        |
|          | <b>References</b>                               | <b>101</b> |

## List of Tables

|   |  |     |
|---|--|-----|
| 1 | Timeline of the work to be done . . . . .                              | 36  |
| 2 | Summary of ways to implement a MySQL-HDFS incremental update . . . . . | 39  |
| 3 | Processing benchmark . . . . .   | 46  |
| 4 | HBase schema, as defined by last year's teams . . . . .                | 48  |
| 5 | Table of acronyms used in this report . . . . .                        | 100 |

## List of Figures

|    |   |    |
|----|---|----|
| 1  | Dataflow diagram of the IDEAL infrastructure [5]  | 11 |
| 2  | Dataflow diagram for incremental update   | 12 |
| 3  | Example structure of a raw tweet record [6]   | 13 |
| 4  | Basic database backup overview [7]  | 22 |
| 5  | Full backup of data each time [8]   | 23 |
| 6  | Simple example to show full data backup   | 23 |
| 7  | Incremental updates [9]   | 24 |
| 8  | Differential backup overview [10]   | 25 |
| 9  | Comparing the three main types of backup options [11]   | 26 |
| 10 | Example of spelling errors, ad-hoc abbreviations and improper casing in a chat record [12]  | 28 |
| 11 | Social network formed amongst users of a social media website [13]  | 30 |
| 12 | Social network based on bibliometrics for media economics [14]  | 31 |
| 13 | Visualization of an entire collection from the previous group [15]  | 33 |
| 14 | Cartoon illustrating the basic principle of PageRank. The size of each face is proportional to the total size of the other faces which are pointing to it. [16] | 34 |
| 15 | Archive tweets and also save in file  | 40 |
| 16 | pt-archiver transfer statistics   | 41 |
| 17 | pt-archiver not able to handle newline characters causing it to break into a newline  | 42 |
| 18 | Hidden tab delimiters in the text file  | 42 |
| 19 | Avro files schema   | 44 |
| 20 | Tweet processing pipeline   | 47 |
| 21 | Potential asynchronous processing pipeline  | 47 |
| 22 | “tweet” column family   | 50 |
| 23 | “clean-tweet” column family   | 51 |
| 24 | “webpage” column family   | 52 |
| 25 | “doc-type” column family  | 52 |
| 26 | “tweet-topic” column family   | 52 |
| 27 | “tweet-cluster” column family   | 53 |
| 28 | Workflow for building social network  | 53 |
| 29 | The format of the tweets we collected from HBase  | 53 |
| 30 | Examples of the tweets we collected from HBase  | 54 |
| 31 | Structure of the social network   | 55 |
| 32 | Visualization of the social network using NetworkX  | 59 |
| 33 | Enlarged bottom part of the graph to see the blue nodes which represent URLs  | 60 |
| 34 | Transfer test SQL data into MySQL table   | 62 |
| 35 | Transfer tweets to the ArchiveDB, and a text file   | 63 |
| 36 | All tweets deleted from the original table, moved to the ArchiveDB, and a separate file   | 64 |
| 37 | Clean the tweets files  | 65 |
| 38 | Different folders on the HDFS system for tweets coming from different tables on the MySQL server  | 65 |
| 39 | Convert a CSV file into Avro file format  | 66 |
| 40 | Two Avro files to be merged on HDFS   | 66 |

|    |   |    |
|----|---|----|
| 41 | Merge the two Avro files on the HDFS file system . . . . .  | 67 |
| 42 | Running procces.sh . . . . .  | 68 |
| 43 | Concatenating part files into a single CSV . . . . .  | 68 |
| 44 | How to use Tweepy to extract useful information from Twitter . . . . .  | 69 |
| 45 | Providing a list of users and gathering data with Tweepy . . . . .  | 70 |
| 46 | The input of the script for building social network . . . . .   | 70 |
| 47 | The output of the script for building social network . . . . .  | 71 |
| 48 | The input of the script for visualization using NetworkX . . . . .  | 71 |
| 49 | The output of the script for visualization using NetworkX . . . . .   | 72 |
| 50 | A relational database model describing a relation between tweets, users, URLs, and<br>other entities [17] . . . . . | 74 |
| 51 | MySQL architecture overview [18] . . . . .  | 75 |
| 52 | Useful commands for MySQL . . . . .   | 76 |
| 53 | High level architecture of Hadoop [19] . . . . .  | 77 |
| 54 | Useful commands for Hadoop . . . . .  | 79 |
| 55 | HDFS architecture overview [20] . . . . .   | 80 |
| 56 | Useful commands for HDFS . . . . .  | 81 |
| 57 | HBase architecture [21] . . . . .   | 82 |
| 58 | Useful commands for HBase . . . . .   | 84 |
| 59 | Examples of importing and exporting data to HDFS using Sqoop . . . . .  | 86 |
| 60 | Workflow of Pig [22] . . . . .  | 87 |
| 61 | Useful commands for Pig . . . . .   | 88 |
| 62 | Simple word count example using Pig [23] . . . . .  | 89 |
| 63 | Installing pt-archiver . . . . .  | 89 |
| 64 | Simple usage of pt-archiver . . . . .   | 90 |
| 65 | Installing csv2avro . . . . .   | 91 |
| 66 | Using csv2avro . . . . .  | 91 |
| 67 | Instructions for installing KVM: Part 1 [24] . . . . .  | 92 |
| 68 | Instructions for installing KVM: Part 2 [24] . . . . .  | 93 |
| 69 | Instructions for installing KVM: Part 3 [24] . . . . .  | 93 |
| 70 | Downloading the Cloudera image for KVM . . . . .  | 94 |
| 71 | Creating a VM using KVM . . . . .   | 94 |
| 72 | Logging into the MySQL database on the Cloudera VM . . . . .  | 95 |
| 73 | Dumping the test SQL database into MySQL . . . . .  | 95 |
| 74 | Example of hard-coded filename combined collection number parameters . . . . .                                      | 97 |
| 75 | The assignment of weights for calculating the user importance factors . . . . .                                     | 98 |

# 1 Requirements

## 1.1 Overview of the IDEAL and GETAR projects

As aforementioned, the work that our team does this semester was to be approached in the context of supporting the IDEAL and GETAR projects.

The IDEAL NSF grant’s abstract describes the project as follows [1]:

“The Integrated Digital Event Archive and Library (IDEAL) system addresses the need for combining the best of digital library and archive technologies in support of stakeholders who are remembering and/or studying important events. It extends the work at Virginia Tech on the Crisis, Tragedy, and Recovery network (see <http://www.ctrnet.net>) to handle government and community events, in addition to a range of significant natural or manmade disasters. It addresses needs of those interested in emergency preparedness/response, digital government, and the social sciences. It proves the effectiveness of the 5S (Societies, Scenarios, Spaces, Structures, Streams) approach to intelligent information systems by crawling and archiving events of broad interest. It leverages and extends the capabilities of the Internet Archive to develop spontaneous event collections that can be permanently archived as well as searched and accessed, and of the Lucid-Works Big Data software that supports scalable indexing, analyzing, and accessing of very large collections.”

Similarly, the GETAR project is described as follows at <http://eventsarchive.org/>

“This project will devise interactive, integrated, digital library/archive systems coupled with linked and expert-curated webpage/tweet collections, covering key parts of the 1997-2020 timeframe, supporting research on urgent global challenge events and initiatives. It will allow diverse stakeholder communities to interactively: collect, organize, browse, visualize, study, analyze, summarize, and explore content and sources related to biodiversity, climate change, crises, disasters, elections, energy policy, environmental policy/planning, geospatial information, green engineering, human rights, inequality, migrations, nuclear power, population growth, resiliency, shootings, sustainability, violence, etc. GETAR will leverage VT research on digital libraries, natural language processing, HCI, information retrieval, machine learning, discovery analytics, and Web archiving.”

For more on these projects, please see <http://eventsarchive.org/>

In order to best serve these projects, we had several major goals to accomplish by the end of the semester. Here, we provide a list of the major project requirements that were crucial to the success of our project.

1. Build an incremental update feature using Apache Sqoop or some other tool to load newly collected tweets from the MySQL repositories that store them into HDFS.
2. Similarly, develop an incremental update feature to load new tweets from HDFS to HBase.

3. Build on the tweet cleaning and noise reduction processes developed by previous teams by augmenting existing Pig scripts to incorporate more cleaning rules.
4. Clean tweets for teams downstream of us, and identify any additional information they would like to see the tweets augmented with.
5. Design, in conjunction with the rest of the class, an HBase table, defining a flexible schema that will allow the class as a whole to incorporate whatever information they would like into our tweet collection.
6. Extract URLs from the tweets to make those available for other teams. This can be done in several ways, including an asynchronous process that can iteratively process the entire tweet collection, or a simple regex extraction.
7. Build a social network of tweets and webpages, and run algorithms on top of this network to define the importance of the network entities.

## 1.2 Current system setup

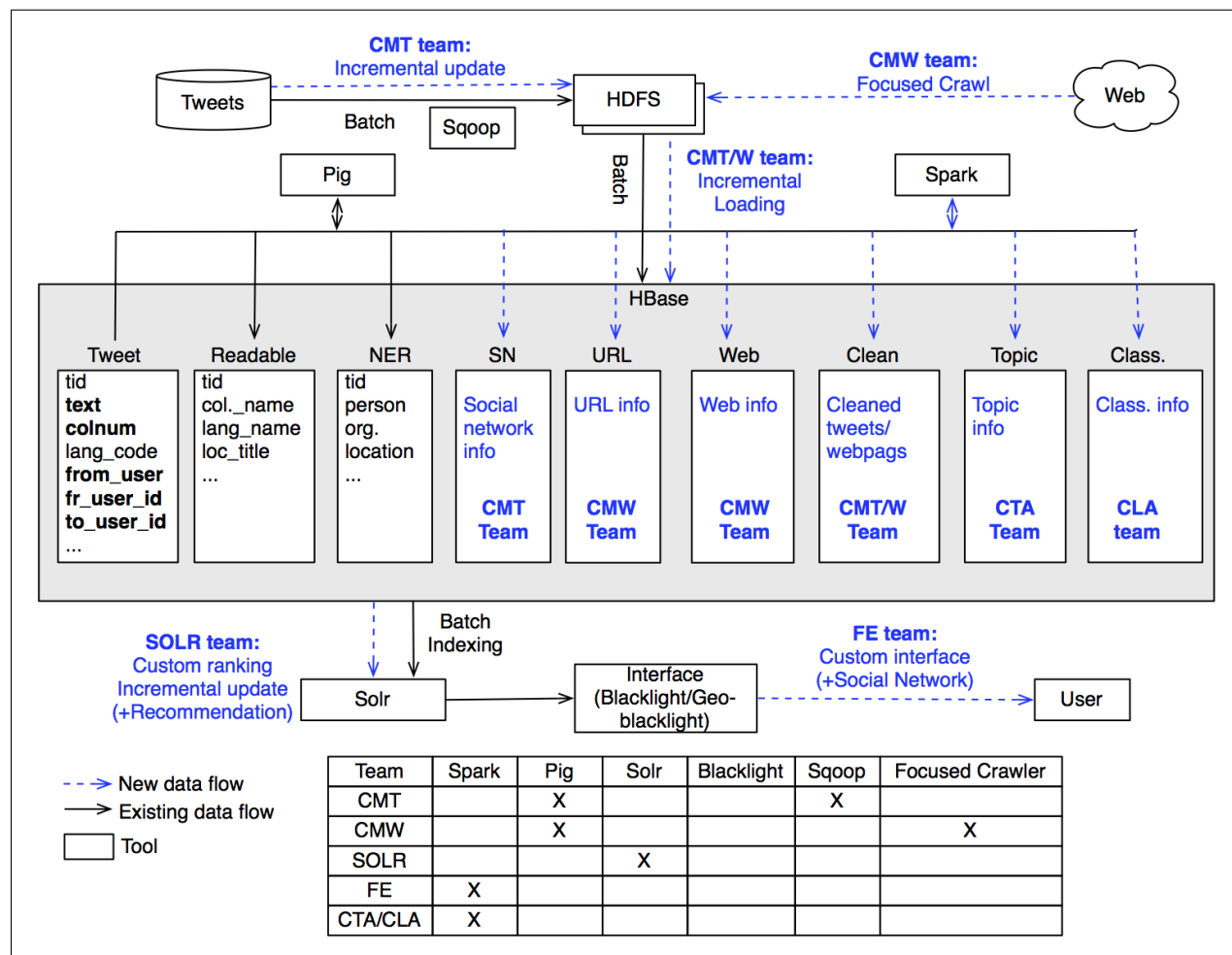


Figure 1: Dataflow diagram of the IDEAL infrastructure [5]

Figure 1 shows the IDEAL/GETAR pipeline as it existed at the beginning of the semester. The parts of the figure that are drawn in black are pieces of the pipeline that had already been implemented; the items in blue are the areas that this semester’s class sought to address.

The pipeline for the system shown above is as follows: tweets are collected by various collection tools (YourTwrapperKeeper [25], Social Feed Manager [26], and the Digital Methods Initiative Twitter Capture and Analysis Toolset [27]) and stored into a MySQL database (this is a simplification, but sufficient detail for this overview). From here, tweets are transported to HDFS using Sqoop. Finally, a batch job to load those tweets into HBase is performed, and other jobs can be run (such as cleaning tweets or fetching additional supplemental information from Twitter).

Once the data has been stored into HBase, Solr is used to index the HBase collection and can be connected to a front end system such as Blacklight [28] so that a GUI can be used to search and

otherwise analyze the collection.

As the CMT team, our work centered around the very beginning of this pipeline, which supported the downstream work of analysis, indexing, and presentation to the user.

### 1.3 Functionality

#### 1.3.1 Incremental database updates

In the IDEAL/GETAR system, two primary databases are used to store the collected tweets, which are curated from various sources. The first is the ArchiveDB, which stores 1.2 billion tweets collected since 2012. The other one is the CollectDB, a MySQL database which stores new tweets collected each day. Various other collections for the IDEAL and GETAR projects encompass some additional 100-million+ tweets, bringing the existing collection's total size to around 1.375 billion tweets (to see the current count, see <http://hadoop.dlib.vt.edu>).

The new tweets we collect each day should be transferred to HDFS and finally into HBase. In the original pipeline, this is done via a Sqoop script, which is able to import all the data from the relational database to HDFS in a batch operation. Unfortunately, this solution is untenable, as there are 1.2+ billion tweets in the ArchiveDB, and it is still growing; we do not want to reprocess it each day we add new data. Therefore, one of our tasks for the semester was the development of an incremental update feature that allows us to obviate the need to reprocess the entire ArchiveDB collection in full each time we wish to update our system with newly harvested tweets. This will yield a huge performance gain over the current batch-processing setup.

After the data is imported from the MySQL databases to HDFS, we transfer the tweet data into HBase. In the original pipeline, that is done via an Apache Pig Latin script. However, this too has been implemented as a batch process, and needed to be re-implemented in an incremental fashion.

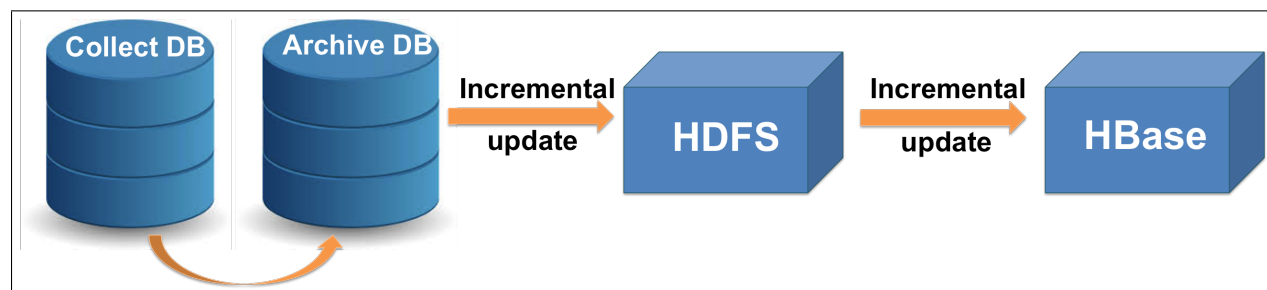


Figure 2: Dataflow diagram for incremental update



### 1.3.2 Tweet cleaning, noise reduction, and augmentation

The tweets that the system collects are raw, unprocessed tweets directly from Twitter’s API. As such, there is a lot of noise in the collection, including things like pornography or spam, that we will be attempting to filter out automatically.

Outside of noise, there is a lot of work to be done to make the tweets usable from an information retrieval perspective, and one of our tasks was to process the tweets into a usable format for downstream teams in our pipeline. This included tasks as lemmatization, the removal of stop words and non-ASCII characters, and other techniques as mentioned in the literature review in section 3.

Finally, we wanted to augment these tweets by adding additional information to the data we have collected. For example, we performed URL extraction and expansion, and appended this information to the tweet record in HBase. Furthermore, we use Stanford’s natural language processing libraries for analyses that include lemmatization and named entity recognition and extraction, storing this information for each record as well [29]. Although we did not get to it in the course of this semester, in the future we would also like to query Twitter for more metadata on older tweets, and update our own databases accordingly.

Figure 3 shows the structure of a raw tweet where all of the useful information like tweet ID, meaningful text, mentions and hashtags are highlighted.

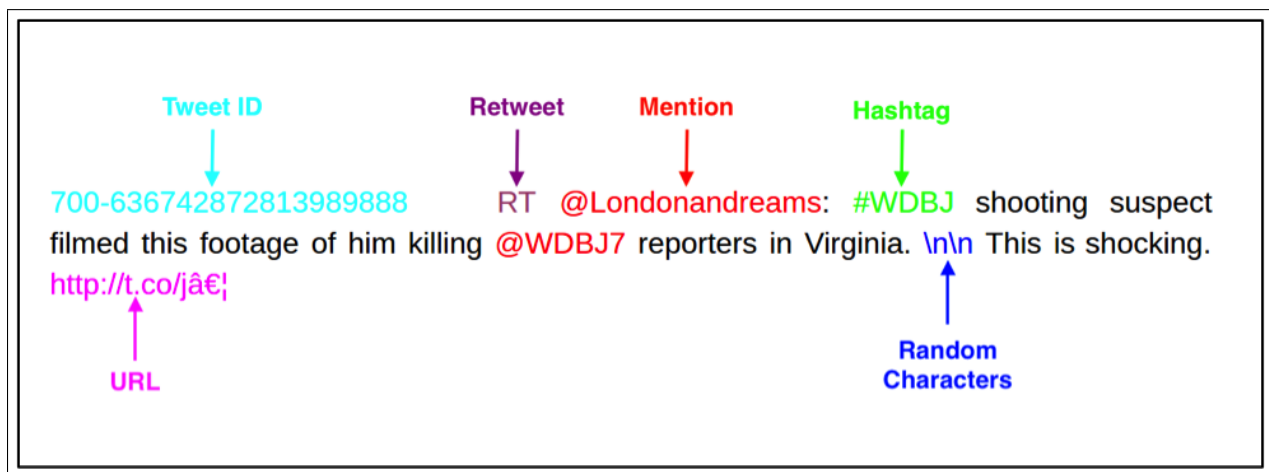


Figure 3: Example structure of a raw tweet record [6]

### 1.3.3 Building a tweet and webpage social network

The idea behind building a social network is to connect every tweet, user, and webpage in a massive graph to model the relationships among those entities. Ultimately, we wanted to be able to attach some kind of importance value to each node in the graph. For example, tweets that have been re-tweeted many times should theoretically be ranked higher than those that have only been re-tweeted a few times or not at all. Tweets from heavily-followed entities like CNN should likewise have a higher importance ranking than individual tweet accounts with a few followers.

There are many considerations that go into building such a graph, such as how one can infer importance (see the literature review in section 3 for an extensive survey of these).

In addition to performing algorithmic analysis on the graph we build, we also developed a preliminary visualization technique to provide system users with a graphical representation of the social networks we build.

## 1.4 Input and output

This section summarizes what data our team receives from other teams and what data we provide in turn.

### 1.4.1 Input

Our team sits at the front of the data pipeline for this project. As input, we take in MySQL databases full of raw and newly-collected tweets harvested from various services.

### 1.4.2 Output

Every other team in the tweet-processing pipeline has downstream responsibilities that have to do with the data that we provide them. We provide an HBase table that stores information on tweets and webpages (one tweet or webpage per row in the table). Each row summarizes the information present in a tweet or a webpage that the row represents, building on the work of previous teams. We have added columns on top of what the team last semester kept track of to provide more utility to the downstream teams.

### 1.4.3 Collaborations

Our team, as the entry point for the system, needed to collaborate with teams further down the pipeline to ensure that they had the information that they need. Specifically:

1. **Front End:** This project required that our team communicate with the Front End team to make sure that we were storing the information that they desire to show to the user. For example, we wanted to know what they would like filtered out in tweets before they present those to individuals using the system for tweet analysis.
2. **Solr:** The most direct interaction we had with the Solr team was through the HBase schema that the class came together to design. It was important that both parties agreed on everything about this schema.

3. **CLA/CTA:** The clustering and topic analysis groups both required tweets in a specific format so that they could perform their respective analyses as effectively as possible. Our team had to communicate with these teams to understand and provide this information.
4. **CMW:** Our team did not have much direct interaction with the CMW team, but as with CLA/CTA, we wanted to provide them with the information that they needed.

## **1.5 Level of performance**

Ideally, our work would enable an incremental update performed daily, during off-peak hours, which would be completed in a timeframe of two hours or fewer.

## **1.6 User support**

We aimed to integrate our solutions seamlessly with the existing system. One thing we were wary of is making changes that were too extensive or trying to incorporate a feature that the rest of the system may not be able to accommodate. This would obviously lead to a diminished user experience resulting from reduced utility of the system, when we could focus attention on other, more pertinent changes that are in line with the existing system and other teams working on the project. For example, if we had made changes to the HBase schema that would affect the tools developed by the previous semester teams to analyze the data stored in the HBase, it would add negative instead of positive value to the entire system.

# **2 Overview of project effort**

## **2.1 Project management**

### **2.1.1 Weekly meetings**

In addition to our regular class meetings, which presented the opportunity to interface with the other teams in our class, we decided to hold two meetings a week to discuss status updates and to work together on developing our project, including meeting impending deadlines and designing our solutions.

### **2.1.2 Communication**

Outside of class, we utilized email and Google Hangouts to coordinate.

### **2.1.3 File-sharing**

To coordinate the sharing of our work, we took advantage of several technologies, including Google Drive and Git (via a GitHub-hosted repository).

## **2.2 Problems and challenges faced**

### **2.2.1 Constraints**

As with any project, there are several constraints on the work that we produce. With specific regard to collection management tweets:

1. Our solution needed to make use of the Hadoop cluster, employing parallel methods wherever possible and adapting the Hadoop ecosystem to our needs.
2. Test collections and evaluation studies needed to be devised to ensure effective and usable operations of the developed systems.
3. Frequent reports and presentations needed to be compiled to keep each project team up-to-date with the current status of each team.
4. Our project needed to be completed in the course of a semester.
5. Maximizing the contributions of every group member and avoiding duplication of effort across our team required us to get better at segmenting work into manageable chunks, working in parallel with one another.

## 2.3 Additional challenges

In addition to the above constraints, we faced the following challenges:

1. Working within and understanding a pre-established ecosystem.
2. The distributed nature of project team responsibilities across the class, requiring a high level of coordination and communication.
3. A number of class items like chapter presentations, chapter reviews, and report reviews require a large amount of time in themselves to complete and needed to be balanced with project work that will advance the state of the system.
4. Incomplete and fragmented documentation from the work of previous semesters. For example, there was scant documentation regarding the last semester's HBase schema, leaving us to figure out what certain columns were used for, and the information that was included was scattered across multiple team reports. Fortunately, we were able to reconstruct this information through the advice of one of the project maintainers, in addition to browsing the tables from last year to see the columns that existed in the database.

## 2.4 Solutions developed

### 2.4.1 First report

Up until our first report, we were mainly focused on understanding the pipeline, how everything is connected in the system, and learning to access online resources provided by Sunshin Lee (one of the current maintainers of the IDEAL/GETAR infrastructure as of the time of writing). A significant portion of our time up to that point went toward reviewing the existing code base and figuring out the project requirements, with the understanding that the majority of our challenges requires in-depth research into the current infrastructure. This included interfacing with those who worked on it before, as well as with the other teams in the class, each of which is working to develop expertise in a particular subsection of the project.

To ameliorate the time constraints imposed by the necessity of reporting and testing, we developed a timeline to keep us on track.

### 2.4.2 Second report

By the time of the second report, our team had a much better grasp on the problems that we were facing, and had done much of the basic research necessary to start formulating solutions.

1. We learned the fundamentals of Hadoop, HDFS, HBase, MySQL databases, Sqoop, Cloudera Search, and some other minor things. A thorough knowledge of these technologies would be needed to successfully complete the project. At the time of the second report, we still had a bit to learn about how these technologies could work together in a pipeline,
2. We set up KVM [24] on one of the lab machines in Room 3050 Torgersen Hall. This machine was more apt for spinning up a virtual machine (VM) since we wanted to allocate enough memory and CPUs to it. Our personal laptops did not have capable-enough hardware.
3. We have assigned a static IP to the lab machine so that everyone in the team can access the VM. Cloudera image “cloudera-quickstart-vm-5.3.0-0-kvm.qcow2” was used as the base images for the VM.
4. Since creation of a social network of tweets is one of our tasks, we spent time reading about the PageRank algorithm and reviewing the work of prior teams with regard to this project. We were still in the process of understanding how we should build this network. We had formulated some rough ideas, but we still had to do more research and anticipated a need for further discussions with Sunshin and Dr. Fox.
5. We looked into the existing tweet cleaning scripts that already exist for this project. We were still trying to figure out if we can/should add more filters in the code to further clean the tweets.

By the end of the work cycle defined by this report, there were still several things that we had yet to define. For example, we now recognized that the class as a whole was severely behind with regards to defining a schema for HBase. One of our sub-goals for the next report was thus embracing a more cohesive, integrated approach to the project, and finalizing specifics like these.

### 2.4.3 Third report

In the time between the second and the third report, we made substantial progress as a group towards several of our main goals. As a class, we navigated the creation of a standard HBase schema for the project. As a group, we began implementation of two of the major components of our work for the semester and were in position to begin the implementation of the third.

1. We created an HBase table on the IDEAL/GETAR infrastructure for the class.
2. Incremental updates between our tweet-collection services onto our HDFS installation were almost finished.
3. We created experimental tweet-cleaning scripts, and were in position to transition to populating HBase tables with those.
4. We developed a plan for the social network, and were about to begin testing and implementation.

For more on each of these endeavors, please see Section 5.

#### 2.4.4 Final report

After our third report, we recognized that we were slightly off track from our self-imposed deadline (please see Section 5 for more details). We began aggressive implementation to finalize our work and integrate it all into a cohesive pipeline. Specifically, we:

1. Finished building the incremental update feature for tweets stored on the MySQL server to the HDFS server. We gave a demo of this functionality to Dr. Fox, Sunshin, and Mohamed. The complete implementation details are included in this report. Collections z\_312 (155000 tweets) and z\_703 (1 million+ tweets) were imported from the MySQL server to the HDFS server in an incremental fashion.
2. Finished building the incremental update feature from HDFS to HBase. In conjunction with the incremental update from MySQL to HDFS, we have formed a coherent ETL pipeline that migrates the tweets from MySQL to HBase and enacts a processing pipeline on them to augment the original records with additional information such as extracted hashtags and entities identified by Stanford's NLP library [29].
3. Processed, augmented, and incrementally uploaded the z\_312 collection from HDFS into our class's HBase table (ideal-cs5604f16). In addition, we also processed and uploaded some collections already on HDFS to HBase. These collections included z\_1, z\_3, z\_20,21,23, z\_24, and z\_312.
4. Finished designing the social network we wished to build from the collection, built a Python program to build the network, and run the pipeline on a subset of tweets as a demonstration. We used the z\_3 collection for this purpose.
5. Delivered a final presentation of our work to the class.
6. Uploaded the entire code base to VTechWorks [30] along with the final project report.

## 2.5 Future work

While we have achieved success in delivering the core requirements of our project, including the incremental updates, tweet processing pipeline, and social network, there are of course some enhancements that could be added on top of our work, including a few minor items that we could not get to because of time constraints.

- **Expanding the range of acceptable characters in tweets:** A part of this pipeline involved the removal of commas, double quotes, and non-ASCII characters from the text of the tweets. We also had to replace null values with zeros in order to successfully convert the CSV files into Avro files using `csv2avro` [31]. In future, we would like to have a mechanism in which we do not have to do these insertions/deletions and preserve the text exactly as it was pulled from the Twitter API.
- **Convert all the MyISAM tables to InnoDB tables:** Currently, all the MySQL tables in the CollectDB and the ArchiveDB are using the MyISAM engine. These tables should be converted to use the InnoDB engine. InnoDB engine is proven to perform better than MyISAM engine [32]. One good source that we found that talks about this conversion can be found in [33].
- **More refined profanity scrubbing:** While we have delivered a basic profanity-censoring solution, it is not very flexible, simply scanning the text for instances of profane words and replacing them, whether they be in URLs, hashtags, or the body of the tweet itself. In the future, it would be convenient to remove the entire hashtag or URL if there is profanity in them.
- **HDFS to HBase pipeline optimization:** We believe that the tweet processing pipeline can be sped up significantly by increasing parallelization (the Java lemmatization and entity recognition step was implemented serially as a proof-of-concept) and moving towards Apache Spark for faster in-memory processing. For more on this, please see Section 5. Another point of optimization would be a better design of the HBase schema as discussed in [34].
- **Remove hard-coded values:** Some values in our codebase, such as the names/locations of files, are hard-coded to suit our own system. In the future, we would like to make these more robust and flexible via script arguments.
- **Implementation of asynchronous tweet augmentation programs:** Unfortunately, the two APIs that we were looking into from Google and Twitter (for geolocation services and additional tweet information respectively) are severely rate-limited in the context of the size of our collection. In the future, we would like to investigate ways to get around or work within these rate limits.
- **Including hashtags in the social network:** We believe it would be beneficial to visualize the relation of tweets, users, and webpages to hashtags in addition to the relations that we have already defined for the former three entities.
- **More advanced social network analysis:** The equations we used for the social network were ad-hoc, and the values we used were based on intuition rather than hard data. In the end, we did not have time to run a more complex algorithm like PageRank, but we would like

to incorporate that algorithm and others like it moving forward. Additionally, we would like to ensure that we utilize importance value calculations that are comparable between different kinds of entities.



### 3 Literature review

This project entailed several components and touched on many concepts. To get a grasp on these, our team reviewed the available literature in several areas. These include:

- Incremental Database Updates
- Noise Reduction
- Large Twitter-focused Social Networks

For a full explanation of the tasks we aimed to accomplish, please see Section 3: Requirements. Further elaboration on the technologies that we utilized for this project can be found in Sections 7.

#### 3.1 Database updates and backups

For any given database, it is important to back it up. Backing up data makes sure that we do not lose any data in cases where the primary database is lost, gets corrupted, or becomes inaccessible for whatever reason. By definition, “Database backup is the process of backing up the operational state, architecture and stored data of database software. It enables the creation of a duplicate instance or copy of a database in case the primary database crashes, is corrupted or is lost.” [35].

There are two primary databases used to collect the tweets in this project, CollectDB and ArchiveDB, in addition to several other smaller databases. Both of these databases are MySQL instances and use the MyISAM engine [36] instead of the more common InnoDB engine [37]. We feel that the choice of MyISAM over InnoDB is not a good one since InnoDB is proven to perform better than MyISAM [32]. MyISAM is better than InnoDB in cases that involve a lot of reading of data from the MySQL tables, but little writing of data into them. This is not the case in our system, where the CollectDB is dynamically updated. As future work, we recommend that someone should look into converting all of the MyISAM tables to InnoDB tables. [33] talks about how this can be done, and though this process is going to be time consuming, we think that this needs to be done sooner than later. We also had a brief conversation about this with Sunshin and he agreed with our suggestion.

CollectDB collects tweets on a dynamic basis using the Twitter API. The ArchiveDB contains the entire set of tweets collected up to the point that the CollectDB was last dumped into the ArchiveDB. Both of these databases are backed up, but the tools that are currently used do not incrementally back-up the data. This led us to do a literature review about the best practices of backing up stored data using any Relation Data Base Management Systems (RDBMS).

The database backups help to safeguard and restore a database. Typically, these backups are performed by some Relation Database Management System (RDBMS). The database administrators can at any given point in time restore the entire database from the backup. The backups are stored on a server that is physically kept away from the main server. The reason to do that is that if there is a natural calamity like an earthquake, it won’t affect both the servers since they will not be in the same physical location (Figure 4 summarizes the process of database backup and subsequent restoration).

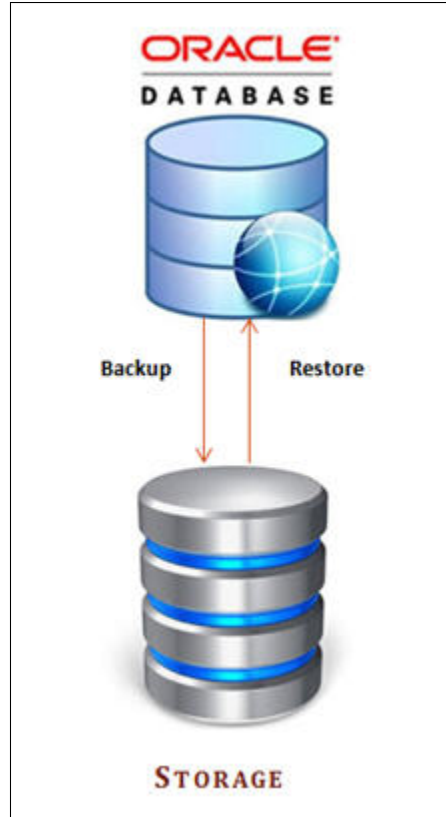


Figure 4: Basic database backup overview [7]

Sometimes, the backups are created to ensure compliance with government regulations so that access to critical business data is not lost in case of a disaster like Hurricane Katrina [38]

Li and Xu discuss the backup mechanisms of the Oracle Database [39]. Some key aspects that they mention are discussed below.

### 3.1.1 Cold database backups

In this type of backup, when the data is being backed up, the original database is closed. This means that during the backup, no active transaction and block change movement can be done. In other words, the original database will not be altered until the backup is done. There are two steps to do a cold backup:

1. Shutdown the database
2. Use the operation system command to copy the files to other locations.

The obvious cost in the case of cold database backups is the cost of being offline. Until data has been backed up, the services remain down. This solution will not work for many use-cases and organizations. For example, for our applications, limited downtime is acceptable, but we want to constantly be collecting tweets. For a heavily-trafficked e-commerce website, a downtime of seconds could mean the loss of millions of dollars.

### 3.1.2 Hot database backups

As the name suggests, in this type of backup, the services running on the main database are not interrupted when the backup happens. In the case of Oracle and MySQL databases, hot backups can be further classified into physical and logical backups, whose difference lies in how the backup is represented (raw file copies versus logical database structures, like the commands used to create the table) [39] [40]. There are several methods of performing these backups, as discussed below.

### 3.1.3 Full backup

In a full backup, the entire database is backed up at regular intervals of time. Every time there is even a slight change in the data set, the entire data set is copied to the backup database. This means copying the original data + the change in data every time. An obvious disadvantage of this approach is the need to copy the same data again and again, essentially wasting time, space, and money.

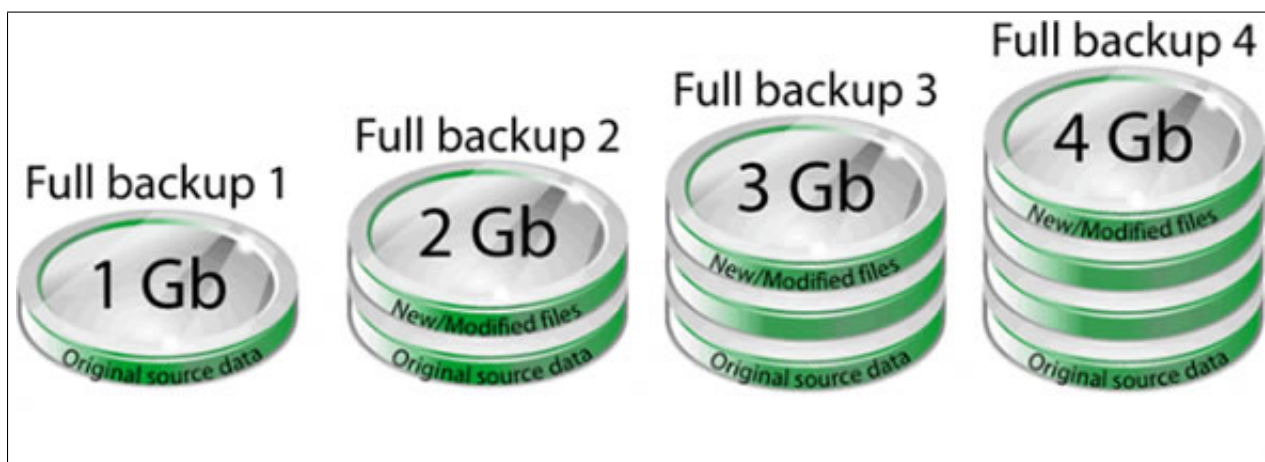


Figure 5: Full backup of data each time [8]

Scheduling of these backups is also important for automation. It is preferred to compress the data before starting the backup procedure to reduce the cost. In certain cases, these backups are encrypted to secure against unauthorized access of data. MySQL does not natively provide these functions. However, the enterprise Backup product can compress InnoDB backups, and also supports encryption. [40]

```
1 | mysqldump --all-databases > dump-$( date '+%Y-%m-%d_%H-%M-%S' ).sql -u root -p
```

Figure 6: Simple example to show full data backup

### 3.1.4 Incremental backup

An incremental backup can also be referred to as an “intelligent” backup. As a matter of fact, only a small percentage of data changes in a database on a regular basis. To backup an entire database even when only a single row has been added to a database is not cost effective. Again, a full backup in such a scenario would entail wasted resources. Figure 7 shows from a high level view how incremental backups work.

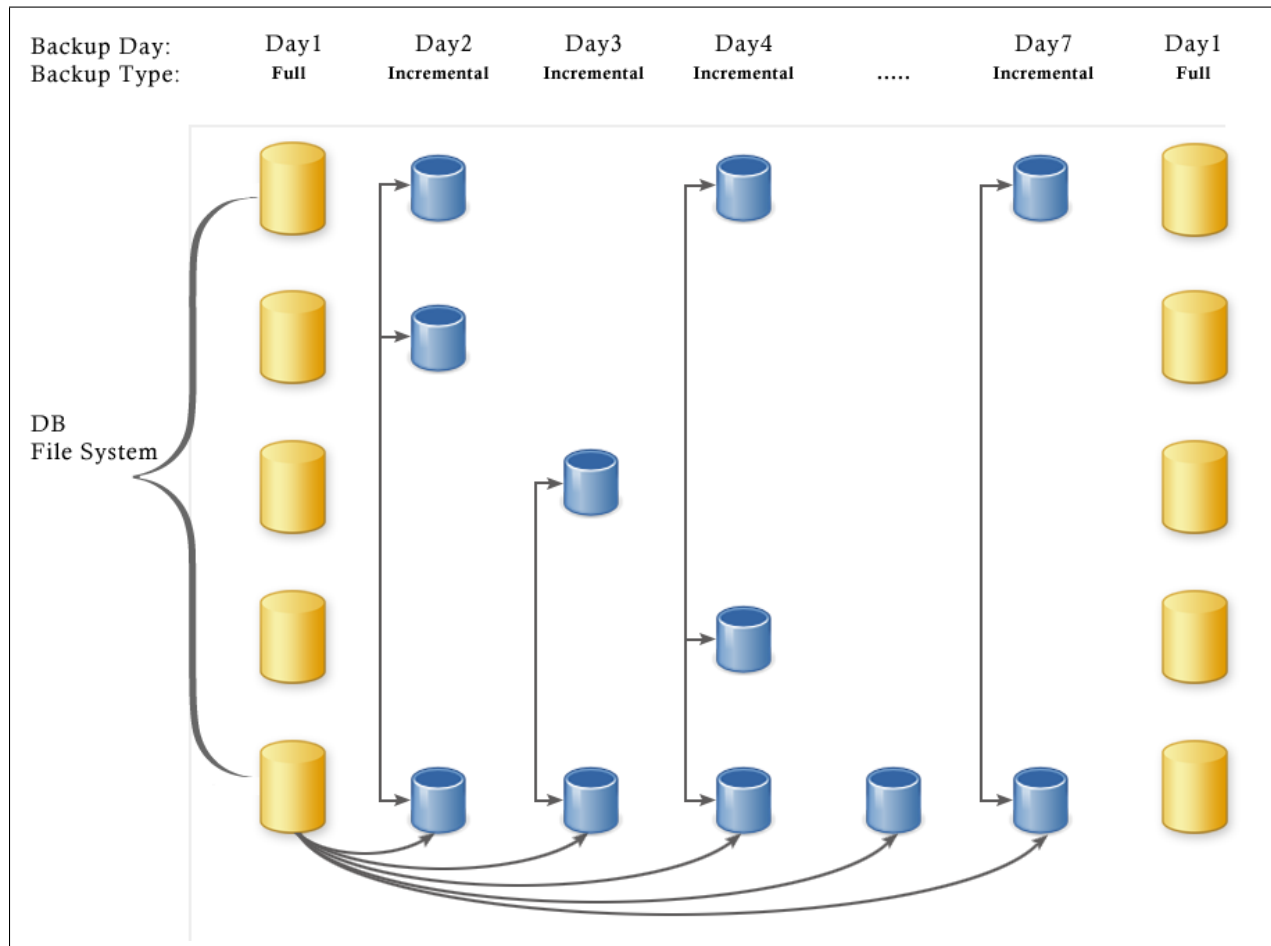


Figure 7: Incremental updates [9]

Incremental updates are initially based on a level 0 backup set. Fewer blocks get written compared to a full backup, and therefore, it is faster and cheaper. Incremental updates can also reduce the recovery time. Physical files are automatically backed up, in contrast to manual copy backup operations.

A disadvantage of using incremental backups is increased restore time. This is because you would need to gather the full backup of the data and then look for increments that have come in since then. So, let us assume that you did a full backup on Monday, and incremental backups on

Tuesday, Wednesday and Thursday. To restore the backup on Friday, we would need all four backup container files - Monday's full backup plus the incremental backups done on Tuesday, Wednesday, and Thursday. Comparing this approach to differential backup (discussed in the next section), we will see that the restore time in differential backup is less.

### 3.1.5 Differential backup

A differential backup copies all the data that has been changed since the last full backup. Differential backup preserves the data, and saves only the difference in the data since the time the last full backup was done [41].

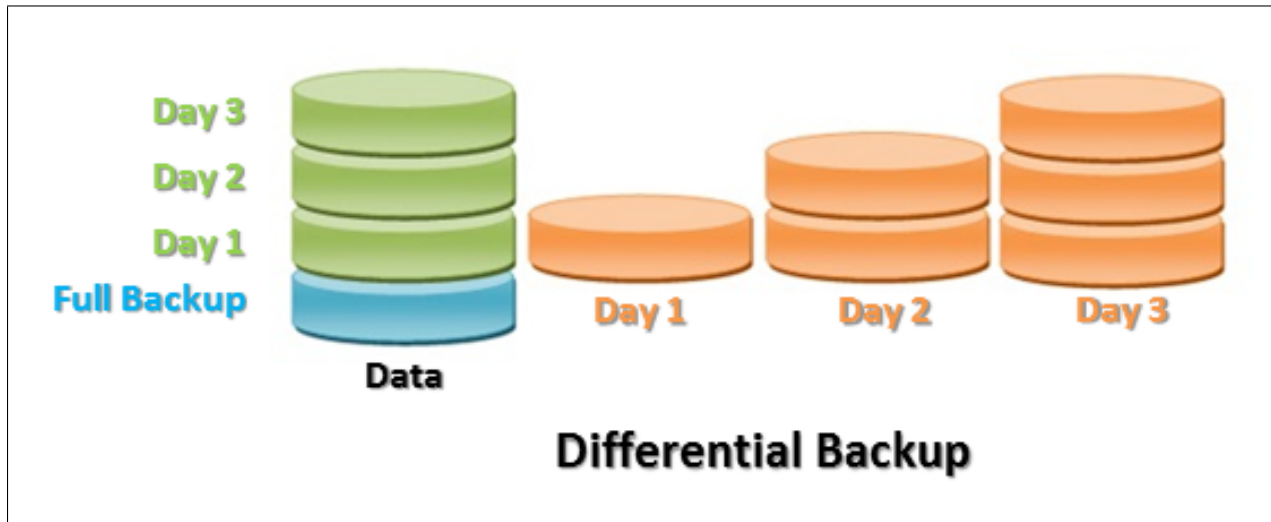


Figure 8: Differential backup overview [10]

1. Pros
  - (a) The process of backing up data is much quicker as it only takes a copy of what has changed.
  - (b) The copy of the backup takes less storage space compared to when a full copy is created.
2. Cons
  - (a) The size of the data differences grows with every backup. If there are too many cycles before a full backup is done, at the end of it, the size of the archive can become very big, making the entire process lengthy.

To give a quick overview of how the three types of backups work, please refer to Figure 9.

| Type/Backup number | Full     | Incremental           | Differential          |
|--------------------|----------|-----------------------|-----------------------|
| Backup 1           | All data | --                    | --                    |
| Backup 2           | All data | Changes from backup 1 | Changes from backup 1 |
| Backup 3           | All data | Changes from backup 2 | Changes from backup 1 |
| Backup 4           | All data | Changes from backup 3 | Changes from backup 1 |

Figure 9: Comparing the three main types of backup options [11]

In our current IDEAL/GETAR system, we do not do incremental or differential updates. There are trade-offs to choosing either of them. Based on our research, and knowing the needs and the structure of the current system, we would recommend that differential update features be incorporated into the current system. This can be future work, but we think it is important to add this feature to the current pipeline soon.

## 3.2 Incremental update from relational database to HDFS

The Collection Management group in the Spring 2016 semester researched the possibility of using Sqoop’s incremental import feature. By using the “lastmodified” mode, which requires a column storing a data value about when each row was last updated, Sqoop will only import rows that were updated after the specific date. The column should be set to the current time with every new inserted row and an update to an existing row, so a row that does not have a modified column will not be imported [6].

### 3.2.1 Percona Toolkit

Percona toolkit [42] is a useful tool to perform various MySQL and system tasks. It provides a collection of advanced command-line tools for user tasks that are complicated and time consuming to perform manually.

Percona toolkit is open-source. It was derived from the Maatkit [43] and Aspersa toolkits [44], neither of which is under active development as of the time of writing. Percona toolkit scripts are professionally developed, tested, and nicely documented. In spite of being open source, the company behind the Percona toolkit also provides an enterprise version of the same. In the enterprise version, they provide a variety of support to the end user.

Currently, the Percona toolkit provides 32 different command line tools that can be used on MySQL databases. Each tool has its own purpose. The tool that is used in our current system is called pt-archiver [45]. This tool archives rows from one MySQL table to another MySQL table.

In our case, it archives rows from the CollectDB to the ArchiveDB. The goal of pt-archiver is to provide a low-impact job that pushes old data from one table to another table “without impacting OLTP queries much” [45]. It can also archive rows to a table hosted on another database on another server.

We can also extend pt-archiver via a plugin mechanism if needed. We can inject our own code that can be useful in applying complex business rules, or building a custom data warehouse. An important thing to note as mentioned on the Percona’s website is [45]:

“pt-archiver does not check for error [sic] when it commits transactions. Commits on PXC can fail, but the tool does not yet check for or retry the transaction when this happens. If it happens, the tool will die.”

Pt-archiver also does error handling in a graceful manner. So, if a user sends a SIGTERM while the pt-archiver process is running, it will terminate the execution, and skip the optimize/analyze phase.

### 3.3 Incremental update from HDFS to HBase

To make the incremental update process automatic, the Collection Management group from the last semester used a cron job scheduler. This allows them to run their Pig scripts periodically. Cron be configured such that data can be loaded onto HDFS and further imported into HBase automatically, each step occurring in its own time window [6]. This eliminates the need for manually performing these update functions.

### 3.4 Text cleaning and noise reduction

The first steps in any text analysis are *text cleaning*, which involves standardization of our input data, and *noise reduction*, which entails making adjustments to the content of the text itself.

#### 3.4.1 Text cleaning

The specific steps taken to clean and normalize text depend on the analysis one means to apply to it. For text cleaning, Burton DeWilde [46] posted a blog, which talked about several things we can do to clean dirty text. A decent, general-purpose cleaning procedure: a) removes digits, non-ASCII characters, URLs, and HTML markup; b) standardizes white space and line breaks; and c) converts all text to lowercase.

#### 3.4.2 Noise reduction

The quality of texts from online sources for ontology engineering can vary anywhere between dirty and clean. On the one hand, the quality of texts in the form of blogs, emails and chat logs, and tweets can be extremely poor. The sentences in dirty texts are typically full of spelling errors, ad-hoc abbreviations and improper casing. On the other hand, clean sources are typically prepared and conformed to high standards. Examples of the latter include academic journals and scientific publications. Texts of different quality will require different treatments during the pre-processing phase, and dirty texts can be much more demanding to handle [12].

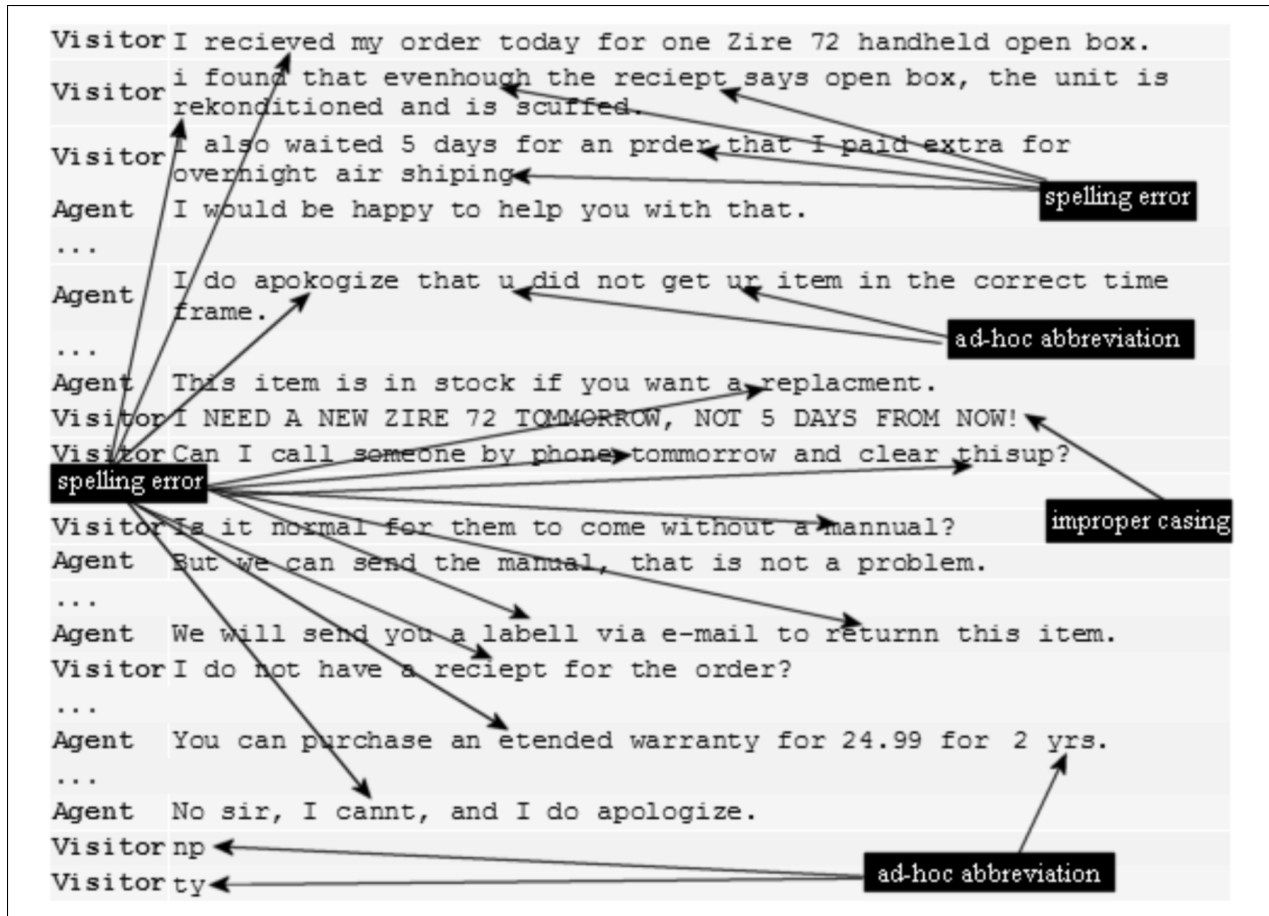


Figure 10: Example of spelling errors, ad-hoc abbreviations and improper casing in a chat record [12]

There are three major types of noise in the dirty text that one needs to handle to allow for further analysis. Figure 10 highlights the various spelling errors, ad-hoc abbreviations and improper casing that occur much more frequently in chat records than in clean texts.

### 1. Spelling detection and correction

The first kind of noise is *spelling errors*. We need to detect spelling errors and correct or account for them. For example, the misspelling of “teh” as “the” is a common typing mistake: a system should be robust enough to recognize that the user likely means “the.” Depending on the context (for example, live interaction), the system might also want to suggest to the user the alternative spelling.

More information is usually required to select a correct replacement from a list of suggestions. Two of the most studied classes of techniques are minimum edit distance and similarity key. The minimum edit distance is the minimal number of insertions, deletions, substitutions, and transpositions needed to transform one string into the other [47, 48].

For example, to change “wear” to “beard” will require a minimum of two operations, namely, a substitution of ‘w’ with ‘b’, and an insertion of ‘d’. The second class of techniques is the



similarity key. The main idea behind it is to map every string into a key such that similarly spelled strings will have identical keys. Hence, the key, computed for each spelling error, will act as a pointer to all similarly spelled words in the dictionary [49].

## 2. Abbreviation expansion

Another type of noise is that of *abbreviations*. We need to recognize shorter forms of words (e.g., “abbr.” or “abbrev.” ), acronyms (e.g., “NATO” ) and initialisms (e.g., “HTML”, “FBI”), and expand them to their corresponding words. The work on detecting and expanding abbreviations is mostly conducted in the realm of named-entity recognition and word-sense disambiguation. There are many approaches to deal with this problem. The approach proposed by [50] begins with the extraction of all abbreviations and definition candidates based on the adjacency to parentheses. The algorithm proposed by [51] is based on rules and heuristics for extracting definitions for abbreviations from texts.

## 3. Case restoration

The third kind of noise is *word case heterogeneity*. We need to detect or account for improper/atypical casing in words and restore/recognize potential alternative casings. For example, if a user types the name “jones” into a search engine, it is likely that they are looking for information on individuals with the name “Jones,” which is naturally capitalized in English.

The approach proposed by [52] identifies sentence boundaries, disambiguates capitalized words, and identifies abbreviations using a list of common words and a list of the most frequent words appearing in sentence-starting positions.

In our project, in order to process the raw tweets for further analysis, we could make use of any of the methods above. In the our class textbook, we found Chapter 2 is useful [53]. It describes methods of word segmentation, true casing, and language detection in documents. According to the report from last semester, the collection management group only cleaned the tweets by removing non-ASCII characters, extracted hashtags, mentions, and URLs from tweet text. Figure 3 gives an example of this. However, based on the demand from the rest of the class, we made modifications to these processes, increasing the number of fields extracted from tweets and adding additional noise reduction processes.

### 3.5 Social networks

Figure 11 shows a small social network formed amongst users of a social media website. Each node represents a different user, and the edges represent the interaction between two users. One can define what it means for two nodes to interact differently in the context of different social media.

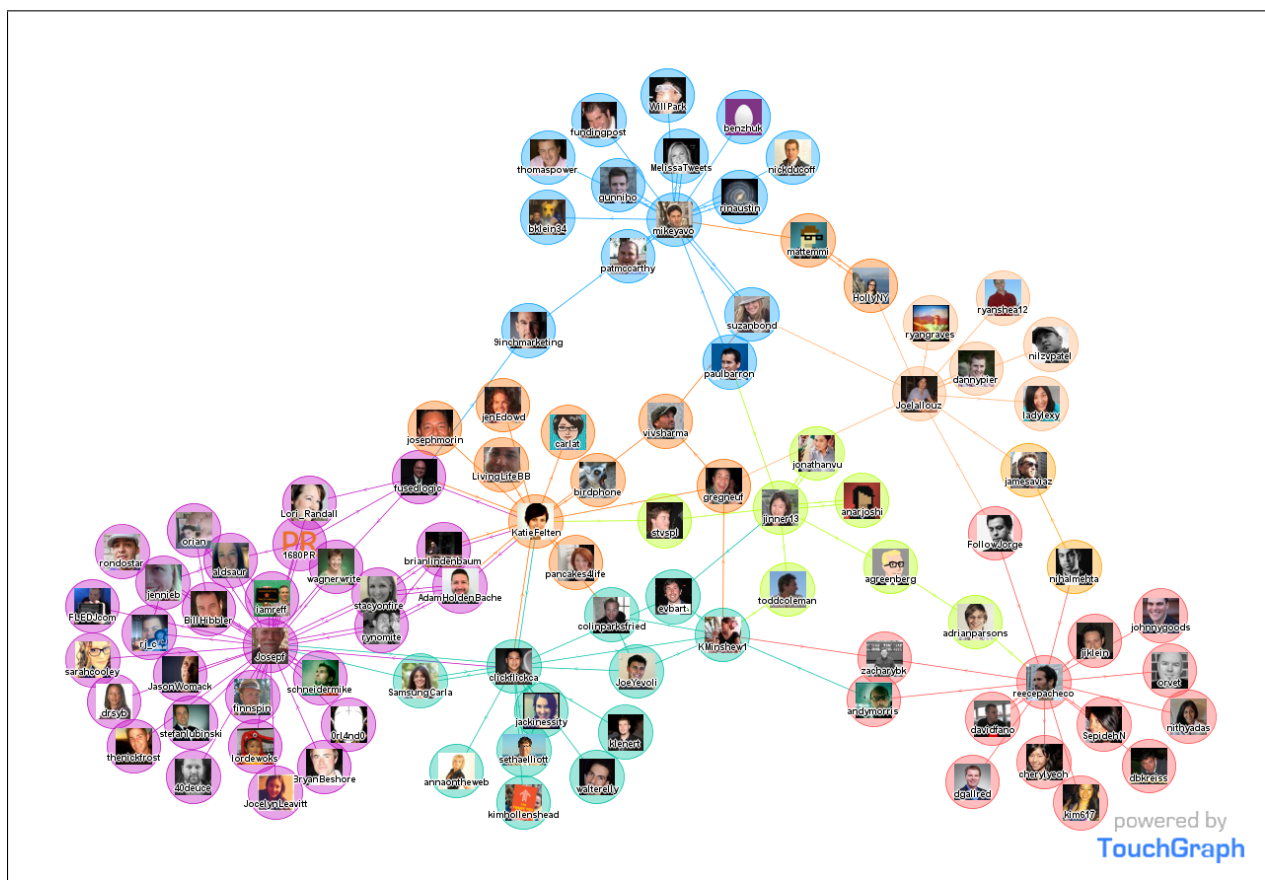


Figure 11: Social network formed amongst users of a social media website [13]

A social network is formally defined as a set of social actors, or nodes, that are connected by one or more types of relations. Nodes, or network members, are the units that are connected by the relations whose patterns researchers study. The units are most commonly individuals, groups or organizations, but in principle any units that can be connected to other units can be studied as nodes, such as webpages, blogs, emails, instant messages, families, journal articles, neighborhoods, classes, sectors within organizations, positions, or nations. Research in a number of academic fields has shown that social networks operate on many levels, from families up to the level of nations, and play a critical role in determining the way problems are solved, organizations are run, and the degree to which individuals succeed in achieving their goals [14].

Social media include all the ways people connect to people through computation. Mobile devices, social networks, email, texting, micro-blogging, and location sharing are just a few of the many ways

people engage in computer-mediated collective action. As people link, like, follow, friend, reply, retweet, comment, tag, rate, review, edit, update, and text one another (among other channels) they form collections of connections. These collections contain network structures that can be extracted, analyzed and visualized. The result can be insights into the structure, size, and key positions in these networks [54].

Figure 12: Social network based on bibliometrics for media economics [14]

The nodes in a social network are not restricted to people or users; we can create social networks for any kind of data where there is relational information between nodes. Figure 12 shows a complex graph regarding the co-citation network in the research field of media economics [12]. This graph makes it easy to see that documents such as Picard 89, Albarran 96, OwenWildman 92, Scherer 7390, Litman 79, Lacy 89, Bagdikian 83/00, and so on, occupy more central positions in the network, indicating that they are more important than others. This is reflected by the number of links to these documents.

Reflected by the number of ties in the graph, we could see that there are more links to them

[14].

Social media networks form in Twitter around a wide range of terms. People talk about the news of the day, celebrities, companies, technology, entertainment, and more. As each person uses Twitter they form networks as they follow, reply to, and mention one another. The public can access these connections, either through the text visible in each tweet or by utilizing the Twitter API to request information such as the list of users that follow a specific tweet author. [55].

We can do a lot of analysis on the social networks, and can be very useful for further applications, such as predictions and recommendations. Social network analysis (SNA) is the study of social structure. Social network analysts are interested in how the individual is embedded within a structure and how the structure emerges from the micro-relations between individual parts. Hence, the greatest advantage of SNA is that it considers how the communication network structure of a group shapes individuals' cognition, attitude and behavior. As an approach to social research, SNA displays four features: structural intuition, systematic relational data, graphic images and mathematical or computational models [14].

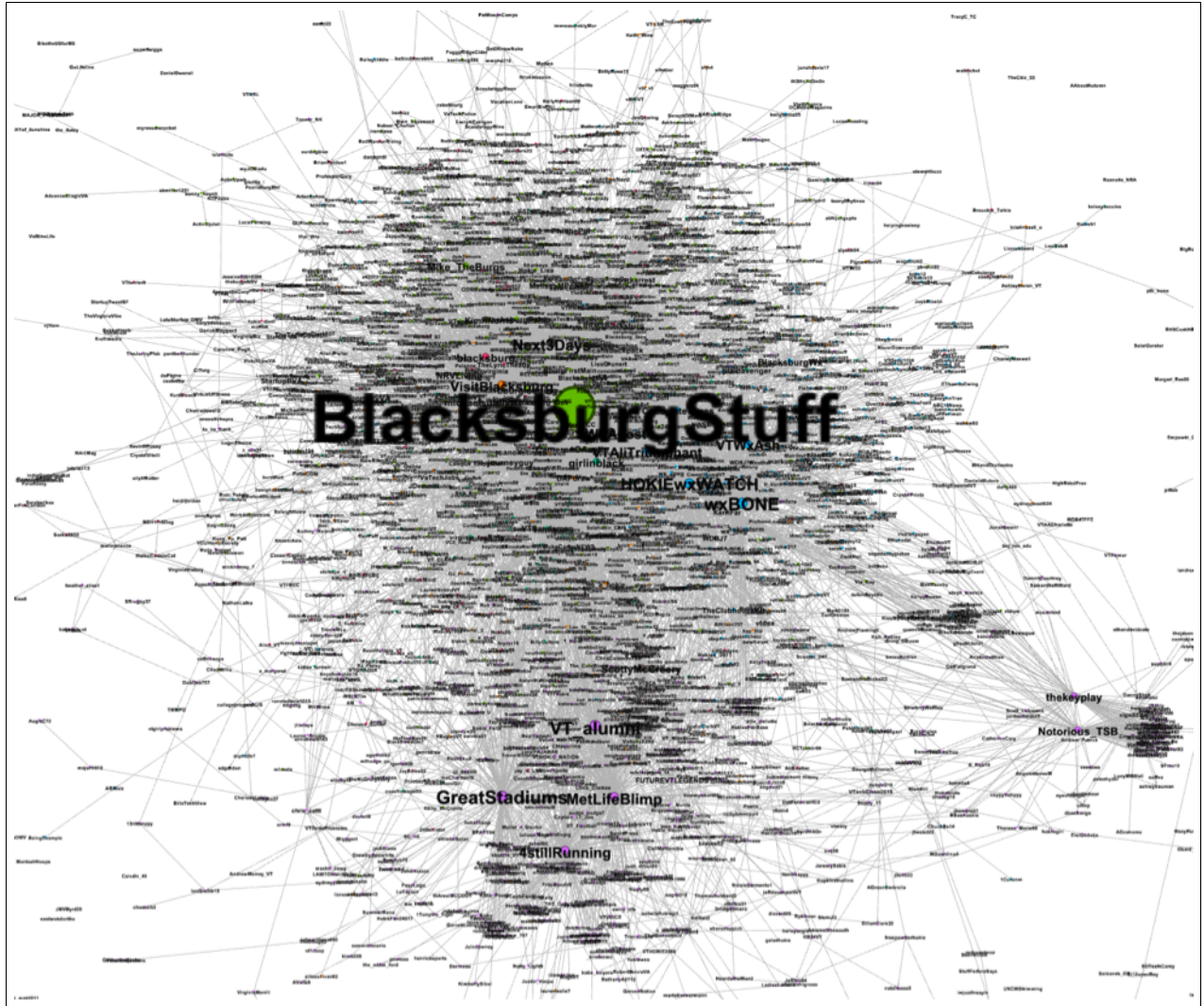


Figure 13: Visualization of an entire collection from the previous group [15]

The cluster and the social network group in the Spring semester built the social network using the accounts in original data as the nodes and the follow information and retweet information as the edges. The result is a matrix of nodes and edges. The weight on the edges is the total number of retweets between the nodes of an edge. They then used this information to produce a visualization, via Gephi, as shown in Figure 13.

### 3.5.1 The PageRank algorithm

PageRank [56] is a famous algorithm that is used by Google to rank webpages in their search engine results. Historically speaking, PageRank is named after Larry Page, co-founder of Google. According to Google [57]:

PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that



more important websites are likely to receive more links from other websites.

PageRank assigns a numerical weighting to each element of a hyperlinked set of documents. The main goal is to measure the relative importance within the set. In the algorithm's original application, webpages are taken as nodes, and hyperlinks as edges. The rank value indicates how important a particular page is. The PageRank of a page is defined recursively, and depends on the number and metric of all pages that link to it.

A lot of research has gone into the PageRank algorithm since the original paper published by Page and Brin [58]. A key goal is to find an effective way of identifying webpages that have attempted to game PageRank scores by taking advantage of various aspects of the algorithm [16].

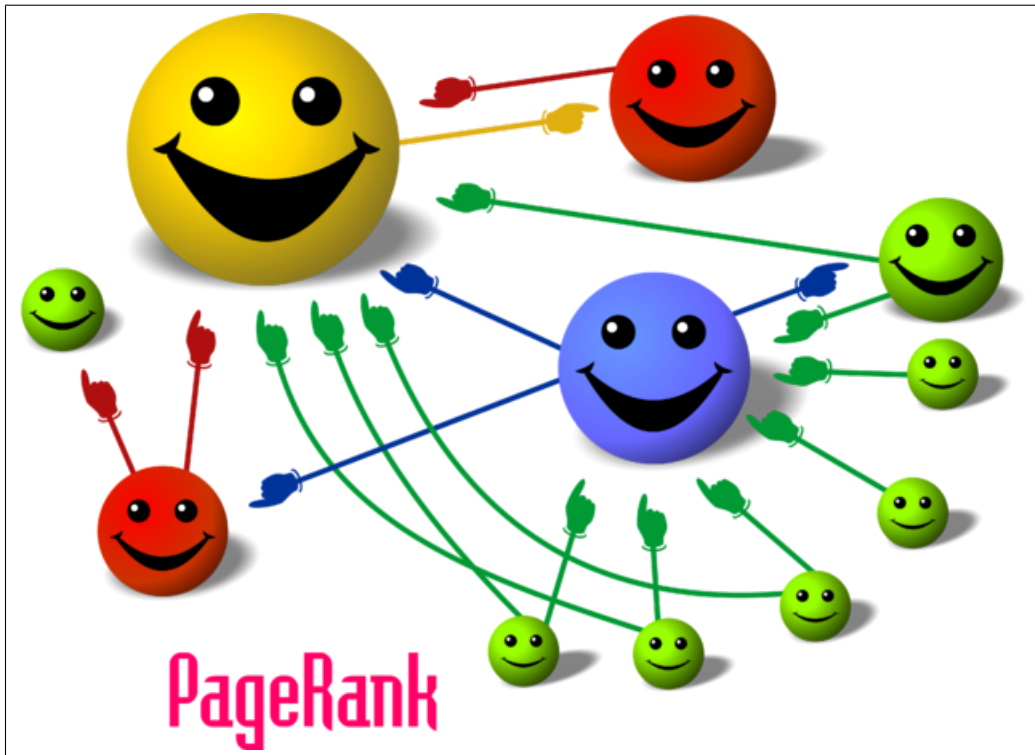


Figure 14: Cartoon illustrating the basic principle of PageRank. The size of each face is proportional to the total size of the other faces which are pointing to it. [16]

## 4 Design

### 4.1 Approach

Our initial approach to this project focused on fully understanding the existing ecosystem of tools and services that comprise the IDEAL and GETAR infrastructure, as integration with that work is a major component of our project. This included meeting with the current system manager as well as examining the documentation written by former project contributors, and having in-class meetings with the other teams in our class.

### 4.2 Tools

Flow diagrams aided our understanding of the existing system allowing us to visualize the connections between the various components. They helped us get a quick grasp on how the current system is structured, how work will be distributed across teams, and how the various software tools and components that we utilize interact with each other.

### 4.3 Methodology

We set up a shared virtual machine for rapid prototyping of Sqoop and Pig scripts on small collections of Twitter data curated for this task by our instructor. Once we were confident in our methods, we transitioned to working on the IDEAL/GETAR cluster.

### 4.4 Conceptual background

From our conversations with our classmates and our instructor, we decided that the most immediate area for us to focus on was moving the tweets into HBase, as this is the basis for any further downstream analysis with this data. In the original IDEAL system this is a two-step process, consisting of harvesting the tweets and storing them into a MySQL database and subsequent transfers from there into HDFS and then into HBase. Our learning and research will mirror the steps that this process entails.

After understanding and implementing the above storage pipeline, our next focus was on cleaning the tweets by filtering out noise like profanity, to make data analysis on the tweets more easier and more useful. Following this, our final efforts were directed toward utilizing the cleaned tweet data to construct various social networks showing the relationships among Twitter users.

To achieve the goals stated in the previous paragraph, we divided up the work and attacked the problem in parallel. While the ultimate pipeline depends on all of these tasks happening sequentially, many of the software components and routines we needed to develop were nicely segmented.

## 5 Implementation

To carry out our tasks, we made extensive use of open-source tools like Apache Pig, Hadoop, csv2avro, avro-tools, NetworkX, Tweepy, and HBase, as well as database technologies like MySQL. For more information on these, please see the included developer’s manual (see Section 7).

### 5.1 Timeline

Table 1: Timeline of the work to be done

| #  | Task   | Deadline        | Status      | Assigned To      |
|----|--|-----------------|-------------|------------------|
| 1  | Chapter presentation   | 9/06/16         | Done        | All              |
| 2  | Research Hadoop and HDFS   | 9/13/16         | Done        | All              |
| 3  | Set up group Cloudera virtual machine  | 9/20/16         | Done        | Faiz             |
| 4  | Research HBase and set up an HBase database                                  | 9/25/16         | Done        | All              |
| 5  | <b>Interim Report 1</b>  | <b>9/20/16</b>  | <b>Done</b> | <b>All</b>       |
| 6  | Create MySQL database on common machine with existing IDEAL MySQL schema     | 9/25/16         | Done        | Faiz, Shuangfei  |
| 8  | Learn Pig and document learning process                                      | 9/25/16         | Done        | Mitch, Shuangfei |
| 9  | Learn how to Load tweets into our MySQL database                             | 9/28/16         | Done        | Faiz, Shuangfei  |
| 10 | Modify MySQL schema to incorporate requests from other teams                 | 9/28/16         | Done        | Mitch, Faiz      |
| 11 | Learn how to load tweets into HDFS using Sqoop                               | 10/01/16        | Done        | Shuangfei, Mitch |
| 12 | Learn how to load tweets from HDFS into HBase using Pig                      | 10/05/16        | Done        | Faiz, Mitch      |
| 13 | Interact with sample data using Pig/Sqoop                                    | 10/07/16        | Done        | Mitch, Faiz      |
| 14 | <b>Interim Report 2</b>  | <b>10/11/16</b> | <b>Done</b> | <b>All</b>       |
| 15 | Develop incremental update of tweets from MySQL to HDFS                      | 10/17/16        | Done        | Faiz, Mitch      |
| 16 | Research improvements to existing scripts for tweet cleaning/noise reduction | 10/24/16        | Done        | Shuangfei, Mitch |



|    |   |                |                           |                  |
|----|---|----------------|---------------------------|------------------|
| 17 | Develop incremental update of tweets from HDFS to HBase         | 10/27/16       | Done                      | Faiz, Mitch      |
| 18 | Custom tweet cleaning & augmentation Pig routine designed       | 10/29/16       | Done                      | Mitch, Faiz      |
| 19 | Research methods for building tweet social networks             | 10/31/16       | Done                      | Shuangfei, Faiz  |
| 20 | <b>Interim Report 3</b>   | <b>11/3/16</b> | <b>Done</b>               | <b>All</b>       |
| 22 | Research methods to identify importance of documents and tweets | 11/3/16        | Done                      | Shuangfei, Mitch |
| 21 | Build tweet social network                                      | 11/18/16       | Done                      | Shuangfei, Faiz  |
| 23 | Run importance algorithm on tweet social network for indexing   | 11/25/16       | Done                      | Shuangfei, Mitch |
| 24 | Integrate developed materials into IDEAL infrastructure         | 11/29/16       | Code submitted for review | All              |
| 25 | <b>Final project presentation</b>                               | <b>12/1/16</b> | <b>Done</b>               | <b>All</b>       |
| 26 | <b>Final report and code submission</b>                         | <b>12/8/16</b> | <b>Done</b>               | <b>All</b>       |

## 5.2 Details on the test data used for this project

- **Test data used:** Dump of the table “z\_312” from the CollectDB was given to us by Sunshin. Later, Sunshin also provided a dump of the table “z\_703” that had more than 1 million tweets.
- **Type of data:** Both the data sets used to test the incremental update feature from MySQL to HDFS were SQL files.
- **Size of the data:** “z\_312” is 58 megabytes in size and “z\_703” is 589 megabytes in size.
- **Number of rows and columns in the data:** “z\_312” has 155657 rows and 14 columns and “z\_703” has 1528869 rows and 14 columns.
- **Number of words in the file:** “z\_312” has 5703802 words and “z\_703” has 55691733 words.
- **Properties of the data:** Uncleaned, raw tweets collected using two Twitter APIs - twitter-stream and twitter-search.
- **Extracted data:** We had to remove non-ASCII characters and misplaced return and newline characters. We also replaced the null characters with zeros.

Once the data was extracted after cleaning it up, we stored each of the 14 columns in HBase [4]. We also extracted more items from the text of the tweet, including hashtags, mentions, and URLs. A more detailed document explaining what was extracted and stored in HBase can be found in Figures 22, 23, 24, 25, 26, and 27.

## 5.3 Deriving our solution

### 5.3.1 Incremental database updates

Our work on this task this semester focused primarily on the two database updates that we were supposed to re-implement in an incremental fashion (MySQL to HDFS and HDFS to HBase).

The central problem surrounding the incremental update for the MySQL to HDFS data transfer was that we needed to somehow be able to keep track of which tweets had already been transferred. We developed several potential solutions to this first problem:

1. We could have added a flag to the MySQL databases to indicate whether or not we have added the rows (tweets) present in the database. New rows would be added with the flag set to false. Then, when new tweets were added to the database, that flag would be set to true. This would allow us to easily keep track of the data that needed to be transferred over to HDFS. Such a setup would allow us to easily employ Sqoop's incremental update feature (see Section 3 for more details).

The disadvantage of this technique is that it would require the modification of the source code of the applications running to collect tweets and populate our collection databases. While one of them is an abandoned project (YourTwapperKeeper [25]), another is in fact undergoing active development and customizing these applications could preclude easy merges of those new updates. Sunshin is also apprehensive about this idea.

2. At present, tweets are added to the system in a batch update that moves everything from the ArchiveDB to HDFS. A second route we explored was simply dumping the CollectDB to the ArchiveDB and HDFS simultaneously, meaning that we would only add the new tweets, not reconstruct the entire HBase table again. At present, the CollectDB is dumped daily to the ArchiveDB via the tool pt-archiver, which performs the transfer to ArchiveDB by locking tables from updates until the transfer of the table is complete. If we could stop the tweet collection services from adding anything to the CollectDB, then in theory, we could dump CollectDB after pt-archiver makes its own updates and simply send the dumped contents over to HDFS. A potential issue would come in interrupting all of our tweet-collection services.
3. A third option we looked at was using pt-archiver to transfer the data into a temporary database with copies of all of the data. This temporary database would be static and would not receive any updates, avoiding issues with stopping our tweet-collection software. From here, we would first transfer all the data to HDFS from this database and then dump the contents to the ArchiveDB, allowing the original CollectDB to continue collecting tweets.

The downside to this solution is that transferring the tweets from the CollectDB to the ArchiveDB is already slow (on the order of 1 hour, according to Sunshin) and this would effectively double the time it would take to move tweets over. Ideally, this process would be initiated daily, during off-hours for system users. Regardless, users of the system utilizing the search interface should not be affected by this technique outside of the time it takes for new tweets to populate the database (something that holds true for all the techniques we mention here). An added benefit is that we would not have to stop collecting the tweets.

4. Another option we explored was to completely ignore the CollectDB and to use a MySQL command like `mysqldbcompare` to generate a list of differences between dumps of the ArchiveDB and then move that data over to HDFS.

The cons of this approach would be that it is likely to be costly (our databases have over 1 billion tweets), but this approach seems fairly straightforward.

5. While researching about `pt-archiver`, we found that `pt-archiver` supports an extra flag called “-file”. Using this flag, we can simultaneously archive the tweets, and save them in a text file. This was good to know since this solution is what we finally implemented. Every time `pt-archiver` archives the tweets, it spews out a text file of the tweets. We made use of this text file of tweets to act as the delta update of tweets.

Table 2 summarizes the different solutions we had to build the MySQL to HDFS incremental update feature along with the potential advantages and disadvantages.

Table 2: Summary of ways to implement a MySQL-HDFS incremental update

| Technique   | Pros  | Cons   |
|---|---|--|
| Adding a flag to MySQL tables   | Very simple to utilize Sqoop’s incremental update feature from this point | Requires modifying the ArchiveDB, and adding a new column  |
| Transfer everything in the CollectDB to the ArchiveDB and HDFS simultaneously | Very simple conceptually  | Requires stopping and re-starting all tweet-collection services  |
| Intermediary database   | Fairly easy to implement  | Slow (requires two or more hours to update)  |
| ArchiveDB diff  | Relatively easier to implement  | Very slow (requires finding the differences between two databases with billions of tweets. Also, requires storing an extra backup of all tweets, which is space-intensive) |

|  |                        |  |
|--|------------------------|--|
| pt-archiver's flag option to create a text file for tweets | Easiest implementation | The text file obtained needs some data cleaning (removal of non-ASCII characters, misplaced newline and return characters, etc.) |
|--|------------------------|--|

Figure 15 shows a screenshot of the command we ran using pt-archiver to spew out a text file of the tweets.

```
[faiz89@viz1 InfoStorage_Project]$ cat transfer_data_using_pt-archiver.bash
#!/bin/bash

cmd="sudo pt-archiver --source h=localhost,D=test_faiz,t=z_312 \
    -u root -p'infostorage' \
    --dest h=localhost,D=test_faiz,t=Archives \
    --file '/home/ubuntu/sample_mysql_data/%Y-%m-%d-%D.%t' \
    --where "1=1" --statistics --ignore"

eval $cmd
retcode=$?
if [ $retcode -ne 0 ]; then
    echo Failed.
else
    echo Success.
fi
```

Figure 15: Archive tweets and also save in file

Running the above script does two things. First, it transfers the tweets from the “z\_312” table in the database named test\_faiz to a table called “Archives”. At the same time, a text file with the name “2016-11-04-test\_faiz.z\_312” gets generated. See Figure 16 for the details.

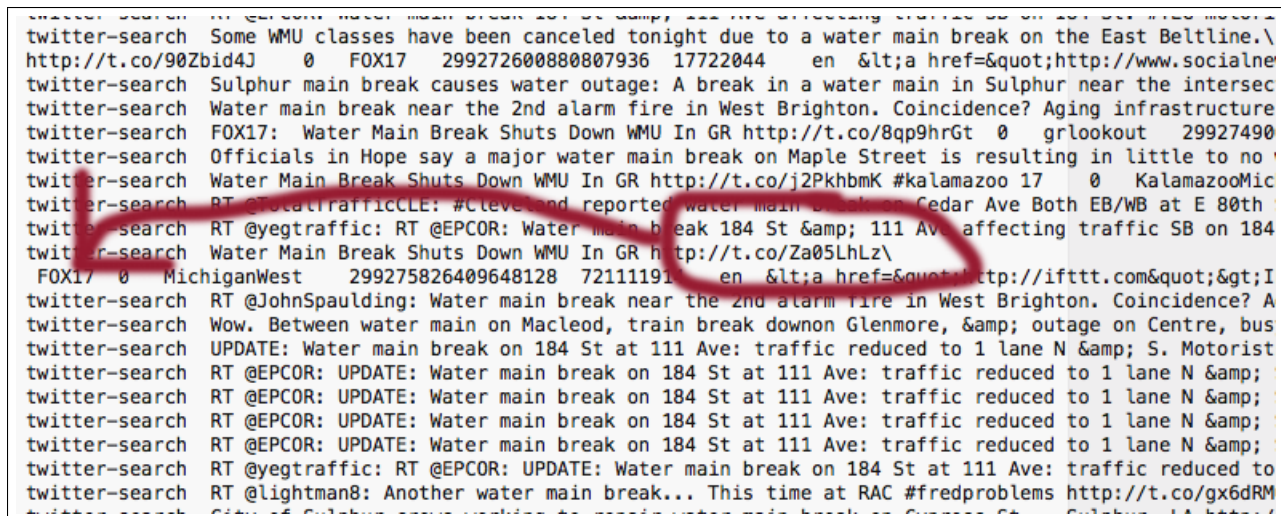
```
[ubuntu@ubuntu InfoStorage_Project]$ ./transfer_data_using_pt-archiver.bash
Started at 2016-11-04T01:40:21, ended at 2016-11-04T01:41:37
Source: D=test_faiz,h=localhost,p=...,t=z_312,u=root
Dest:   D=test_faiz,h=localhost,p=...,t=Archives,u=root
SELECT 155657
INSERT 0
DELETE 155657
Action      Count      Time      Pct
deleting    155657    25.5445   33.53
select      155658    16.6420   21.84
inserting   155657    11.9007   15.62
commit      311316    7.3739    9.68
print_file  155657    0.8382    1.10
other       0         13.8833   18.22
Success.
[ubuntu@ubuntu InfoStorage_Project]$
```

Figure 16: pt-archiver transfer statistics

Once we got the text file, we did not care about the new tweets being added to the ArchiveDB since at this point, we were dealing only with the text file of the tweets created. This text file served as the means to do the incremental update.

An important note about pt-archiver is that while it is archiving rows from one table to another, it locks the source table. This makes sure that the table is not being updated while rows are being archived from it. This is an important feature of pt-archiver as without this feature, building the incremental update feature would have been more complicated.

Our next challenge was to parse this text file, and meaningfully extract the 14 columns in each row. However, the text file we got from pt-archiver was badly structured. For example, a lot of tweets had misplaced return and newline characters. In such cases, pt-archiver starts a newline assuming it is a new row in the MySQL database. Figure 17 shows a screenshot of the problem.



```

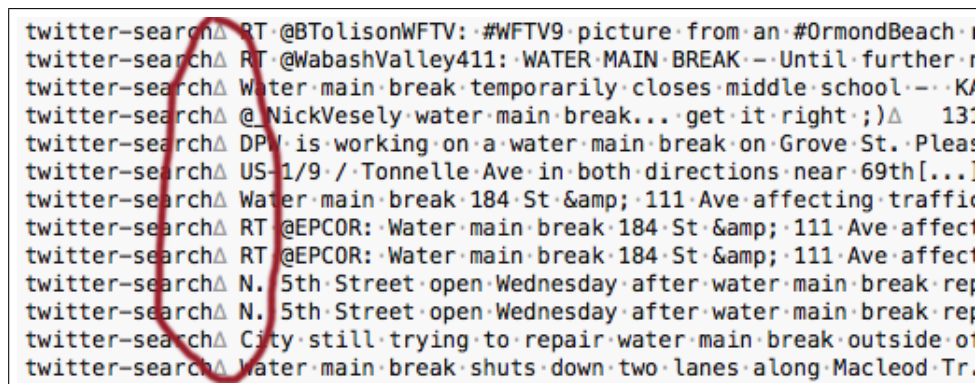
twitter-search Some WMU classes have been canceled tonight due to a water main break on the East Beltline.\
http://t.co/90Zbid4J 0 FOX17 29927260080807936 17722044 en &lt;a href=&quot;http://www.socialne
twitter-search Sulphur main break causes water outage: A break in a water main in Sulphur near the intersec
twitter-search Water main break near the 2nd alarm fire in West Brighton. Coincidence? Aging infrastructure
twitter-search FOX17: Water Main Break Shuts Down WMU In GR http://t.co/8qp9hrGt 0 grlookout 29927490
twitter-search Officials in Hope say a major water main break on Maple Street is resulting in little to no
twitter-search Water Main Break Shuts Down WMU In GR http://t.co/j2PkhbmK #kalamazoo 17 0 KalamazooMic
twitter-search RT @stattrafficCLE: #Cleveland reported water main break on Cedar Ave Both EB/WB at E 80th
twitter-search RT @yegtraffic: RT @EPCOR: Water main break 184 St & 111 Ave affecting traffic SB on 184
twitter-search Water Main Break Shuts Down WMU In GR http://t.co/Za05LhLz\
FOX17 0 MichiganWest 299275826409648128 72111191 en &lt;a href=&quot;http://ifttt.com&quot;&gt;I
twitter-search RT @JohnSpaulding: Water main break near the 2nd alarm fire in West Brighton. Coincidence? A
twitter-search Wow. Between water main on Macleod, train break downon Glenmore, & outage on Centre, bus
twitter-search UPDATE: Water main break on 184 St at 111 Ave: traffic reduced to 1 lane N & S. Motorist
twitter-search RT @EPCOR: UPDATE: Water main break on 184 St at 111 Ave: traffic reduced to 1 lane N &
twitter-search RT @EPCOR: UPDATE: Water main break on 184 St at 111 Ave: traffic reduced to 1 lane N &
twitter-search RT @EPCOR: UPDATE: Water main break on 184 St at 111 Ave: traffic reduced to 1 lane N &
twitter-search RT @yegtraffic: RT @EPCOR: UPDATE: Water main break on 184 St at 111 Ave: traffic reduced to 1 lane N &
twitter-search RT @lightman8: Another water main break... This time at RAC #fredproblems http://t.co/gx6dRM

```

Figure 17: pt-archiver not able to handle newline characters causing it to break into a newline

Every newline needed to start with either “twitter-search” or “twitter-stream” since those are the names of the two APIs used to gather tweets, but in this case, it was starting from “FOX17”. This made it difficult for us to parse the text file and extract all items appropriately. Ideally, each row is supposed to have 14 items, which we wanted to store in a Python list, and then write out to a CSV file. But because of these newline and carriage return characters, each row now did not necessarily have 14 items.

Another peculiar thing to notice about this text file generated by pt-archiver was the use of the tab delimiter. This tab delimiter was effectively hidden in that we could not see it by just concatenating the file. However, when we observed this file under TextWrangler [59], a code editing software, we could see these hidden delimiters. See Figure 18 for the screenshot showing the hidden tabs.



```

twitter-search RT @BTolisonWFTV: #WFTV9 picture from an #OrmondBeach
twitter-search RT @WabashValley411: WATER MAIN BREAK - Until further
twitter-search Water main break temporarily closes middle school - K/
twitter-search @NickVesely water main break... get it right;) 13:
twitter-search DPM is working on a water main break on Grove St. Plea
twitter-search US-1/9 / Tonnelle Ave in both directions near 69th[...]
twitter-search Water main break 184 St & 111 Ave affecting traffic
twitter-search RT @EPCOR: Water main break 184 St & 111 Ave affect
twitter-search RT @EPCOR: Water main break 184 St & 111 Ave affect
twitter-search N. 5th Street open Wednesday after water main break re
twitter-search N. 5th Street open Wednesday after water main break re
twitter-search City still trying to repair water main break outside o
twitter-search Water main break shuts down two lanes along Macleod Tr

```

Figure 18: Hidden tab delimiters in the text file

Our first approach included removing all newline and carriage return symbols from the file using some delimiter in the text file. We wrote a Python script to do that, but that also removed all the newline characters at the end of each line, which we did not want. We wanted a newline character

at the end of each line so that there remain 14 items in each row.

We then tried to write one regular expressions for each of the 14 columns that would parse the documents and separate the entire text based on the regular expressions. There were two issues with this approach - a) this method involved writing 14 unique regular expressions that seemed like an overkill; and b) it was difficult to come up with a perfect regular expression for the “text” field. The raw text field contains a lot of non-ASCII and non-recognizable characters. These characters were not letting us successfully create a generic regular expression for the “text” field.

Finally, what worked for us was writing a simple bash script that did a few things:

1. Remove every newline character. Newline characters can be “\n”, “\r\n”, or “\r”. This effectively merged the entire document into a single line.
2. Insert a newline character before every occurrence of the word “twitter-search” and “twitter-stream”. We did this based on the observation that the first column is always either “twitter-search” or “twitter-stream”.

Once we cleaned the text file of the unnecessary newline and return characters, single and double quotes, and replaced all the null characters with zeros, we converted this text file into a CSV file. Our next job was to convert this CSV file into an Avro file [60]. Avro file format has its advantages we discussed in section 3. Avro files have a schema stored in them along with the data. It took us some time to actually understand what are these Avro files and where and how is the schema defined. A part of the Avro files schema is shown in Figure 19.

```

{
  "type": "record",
  "namespace": "hadoop.dlib.vt.edu",
  "name": "hadoop.dlib.vt.edu.tweets",
  "fields": [{ "name" : "archivesource",
                "type" : "string",
                "default" : "NONE"},
              { "name": "text",
                "type": "string",
                "default" : "NONE"},
              { "name": "to_user_id",
                "type": "string",
                "default" : "NONE"},
              { "name": "from_user",
                "type": "string",
                "default" : "NONE"},
              { "name": "id",
                "type": "string",
                "default" : "NONE"},
              { "name": "from_user_id",
                "type": "string",
                "default" : "NONE"},
              { "name": "iso_language_code",
                "type": "string",
                "default" : "NONE"},
              { "name": "source",
                "type": "string",
                "default" : "NONE"},
              { "name": "profile_image_url",
                "type": "string",
                "default" : "NONE"},

```

Figure 19: Avro files schema

We used `csv2avro` [31], an open source tool, to convert the CSV file to an Avro file. For more details on this tool, please refer to Section 3.

The minimum block size of the HDFS filesystem on the cluster is 256 megabytes. This meant that even a 1 byte file stored on HDFS will consume 256 megabytes of space. In our case, since we were implementing an incremental update feature, the size of these Avro files was small. For our test data (“z\_312” table), it was a 49 megabytes file in size. Hence, 201 megabytes of space would



effectively be wasted. This problem will only get worse as we get more data from the Twitter APIs and we do more incremental updates.

Based on our conversation with Sunshin on November 3 in the class, we decided to look into merging the Avro files. It was not as simple as just concatenating the two CSV files since it also involves the schema. After doing some research and failing to merge the Avro files using PiggyBank[61], we made use of the avro-tools [60] to merge two Avro files.

Thus, we were successful in implementing the incremental update feature of tweets from the MySQL database to the HDFS server. Refer to Section 6 for more details on this.

### 5.3.2 Noise reduction and cleaning

After discussion with the CLA team, we realized that we had a greater role in the actual cleaning of the data than previously thought.

The previous work done by last year’s collection management team seemed to do only very basic cleaning, like the removal of curse words. However, CLA asked us to do analytic processing on the tweets, like lemmatization, in curating a final, clean version of tweet source text. Our discussions with the CTA team revealed that they would like something similar. CLA mentioned that Stanford has some open source tools available for NLP that looked like they might be useful for this task and others (like automatically identifying named entities in text).

Further investigation revealed that Sunshin had several working scripts for these items that he kindly shared with us for the completion of our project, including code that uses the Stanford tools. Unfortunately, these tools were inefficient and did not cover the full extent of functionality that we would ultimately need to provide. We wound up using them as convenient starting points in pursuit of our own solutions.

In the end, we utilized a combination of Apache Pig Latin (per the recommendation of Sunshin), Java, and Python to implement the processing pipeline. This is divided into three stages, each done on the granularity of a single collection:

1. The initial read would be done on a temporary Avro file stored in HDFS as a result of the incremental update from MySQL to HDFS. This read will load the information from the raw tweet into the tweet column family in our class’s HBase table. Additionally, it would initialize various other columns in the clean-tweet column family to an empty string to simplify later processing.
2. The second step of the pipeline utilizes a Java program that draws on Stanford’s CoreNLP suite and the HBase Java API to scan our table for tweets that have not been lemmatized (i.e., tweets where the lemmatized column in the clean-tweet column family is an empty string), and utilizes the CoreNLP code to lemmatize the tweets and extract named entities from them. This step is currently performed serially, and would be a major target for optimization in the future.

3. The final step in this pipeline is additional cleaning and tweet augmentation. We again scan the HBase table for tweets with specific fields that have not been initialized, filtering out the rest. With these remaining tweets, we use a combination of Apache Pig Latin and Python to extract information like hashtags and mentions, and prepare the body of each tweet for the downstream groups (essentially, we populate most of the clean-tweet columns described in Figure 23. This step is somewhat sluggish (please see Table 3 for more details).
4. Once this pipeline has finished, we merge the temporary Avro file with the Avro file archive for the entire collection, as described previously.

The core of this pipeline can be seen in Figure 20.

Table 3: Processing benchmark

|                                       |                        |
|---------------------------------------|------------------------|
| <b>Collection:</b>                    | 312 (Water Main Break) |
| <b>Number of Tweets:</b>              | 155657                 |
| <b>Initial Step (Duration):</b>       | 2 minutes              |
| <b>Lemmatization Step (Duration):</b> | 33 minutes             |
| <b>Cleaning Step (Duration)</b>       | 27 minutes             |

In the future, there are several options that could potentially speed up the processing as described above. These are enumerated below:

1. One could use the Java APIs to read from the Avro file directly rather than scan HBase for valid rows within each column. Scanning HBase still takes a large amount of time, even if we avoid re-processing tweets in the table by filtering some of them out.
2. The Java step is currently run serially: making that a parallel process that works on separate parts of the table would offer a linear speed-up.
3. Another option would be forgoing Java entirely for the lemmatization and cleaning steps: the CLA group utilized Apache Spark [62] for processing in their own work this semester, including lemmatization and the removal of stop words. We refer to their report on VTechWorks for more details on this process. As Spark processes data in-memory, avoiding needless writes and reads to disk, it can offer a significant reduction in overhead.
4. Finally, the use of Python for several of the cleaning steps is likely inimical to program execution speed. Converting to Java or using native Pig functions could potentially be faster.

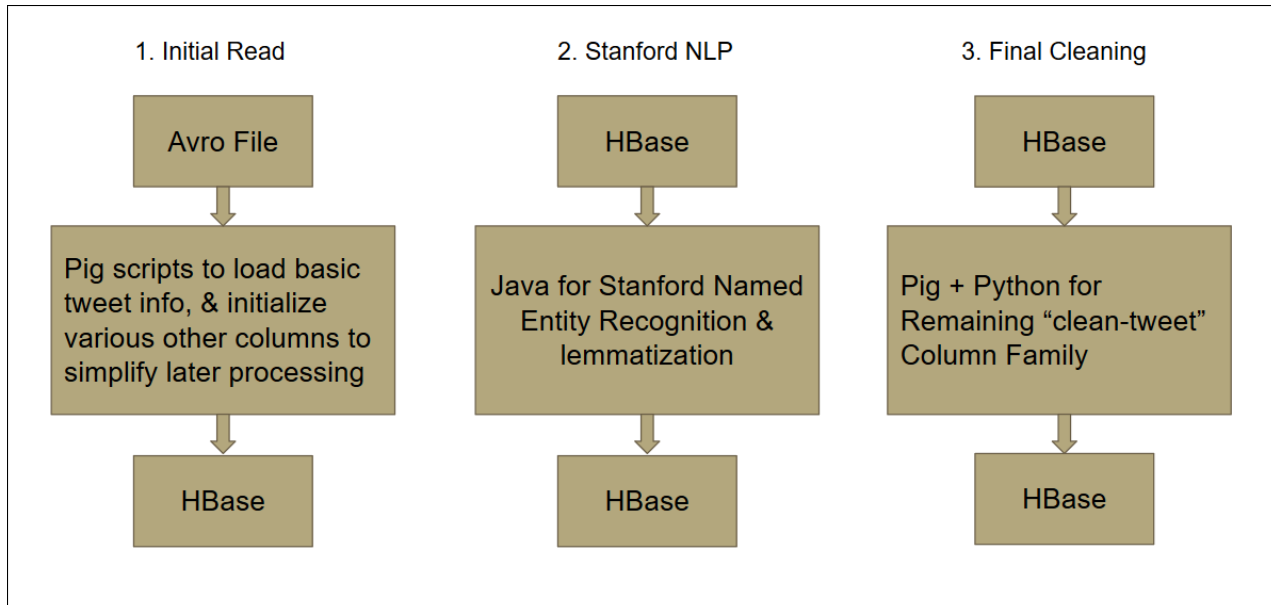


Figure 20: Tweet processing pipeline

Additionally, we developed a conceptual framework to implement asynchronous updates that builds upon the rest of our work, as seen in Figure 21.

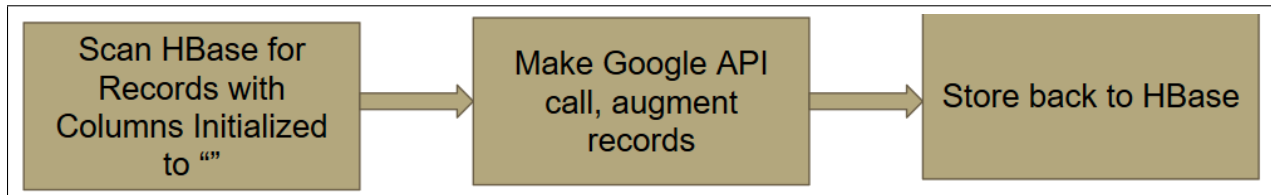


Figure 21: Potential asynchronous processing pipeline

We store multiple versions of cleaned tweet data for the various teams. CLA and CTA, for example, asked for stop word removal, to improve tweet classification and clustering. On the other hand, the FE team does not desire us to remove stop words (a decision informed by our textbook, which states that the trend in modern search engines has been to use fewer and fewer stop words) [53].

One item that we ran out of time to do was conduct meaningful experiments with our solutions.

Table 4: HBase schema, as defined by last year’s teams

| Column Family    | Column               |
|------------------|----------------------|
| clean_tweet      | clean_text           |
| clean_tweet      | collection           |
| clean_tweet      | hashtags             |
| clean_tweet      | urls                 |
| clean_tweet      | collection           |
| clean_tweet      | hashtags             |
| clean_tweet      | urls                 |
| clean_tweet      | mentions             |
| tweet            | archivesource        |
| tweet            | created_at           |
| tweet            | from_user            |
| tweet            | from_user_id         |
| tweet            | geo_coordinates_0    |
| tweet            | geo_coordinates_1    |
| tweet            | iso_language_code    |
| tweet            | profile_image_url    |
| tweet            | source               |
| tweet            | time                 |
| tweet            | to_user_id           |
| tweets_topics    | probability_list     |
| tweets_topics    | topic_label          |
| clean_web        | text_clean_profanity |
| clean_web        | collection           |
| clean_web        | urls                 |
| clean_web        | lang                 |
| clean_web        | domain               |
| clean_web        | title                |
| doctype          | doctype              |
| cf_cf            | sim_scores           |
| cf_cf            | sim_tweets           |
| clustered-tweets | cluster-label        |
| clustered-tweets | doc-probability      |
| classification   | relevance            |

One major issue we encountered in this project is the lack of a coherent schema from last year’s Collection Management team [6]. The above schema, taken from last year’s Solr team, provided us with a basis on which to build a new schema. Please see Figures 22, 23, 24, 25, 26, and 27 to learn more about the schema that we have developed in conjunction with this year’s class.

As a result of our difficulties reconstructing last year’s schema, our team put a great focus on curating comprehensive documentation on the schema our class will develop. We emphasized the importance of this endeavor in our interactions with the rest of the teams.

It is important to note that HBase is a NoSQL data store, with more flexibility than a traditional MySQL database [34]. In effect, you can think of the database as a giant key-value store, where a row is identified by a unique key, which links to a set of column families which each contain any number of columns. While the column families have to be defined up front, the columns can be defined on the fly (hence the importance of documenting a standard schema) [34]. It is not necessary for every row in HBase to have a value for every column family, whereas in a traditional MySQL database, every row has exactly the same number of columns, with some left empty as need be. These sparse datasets can waste a huge amount of space in traditional databases, but HBase is optimized to handle cases like these [34].

The tables below are, mostly, self-explanatory, but the latter three table columns might seem a bit confusing. These are primarily the purview of the Solr and Front End teams, with “stored” and “indexed” being Solr terms and “facets” a part of search interface vocabulary. As such, these columns are mostly outside the scope of the CMT team’s own project.

The row key of each tweet in the table is the tweet’s collection number, concatenated with the tweet’s ID via a hyphen. For a webpage, the key is the page’s URL. For simplicity’s sake, all of the data stored in HBase is stored as a chararray.

| Table           | Column Family | Column            | Description                            | Example   | Stored | Indexed          | Facet |
|-----------------|---------------|-------------------|--|---|--------|------------------|-------|
| ideal-cs5604f16 | tweet         | colnum            | number of the collection               | 651   | No     | N/A              | No    |
|                 |               | tweet_id          | tweet's unique identifier              | 299755872668758016  | No     | N/A              | No    |
|                 |               | archivesource     | twitter API's type                     | twitter-search, twitter-stream  | Yes    | archive_source_s | No    |
|                 |               | source            | platform's type                        | <a href="http://www.facebook.com/twitter" rel="nofollow">Facebook</a><br><a href="http://twitterfeed.com" rel="nofollow">twitterfeed</a><br><a href="http://dlvr.it" rel="nofollow">dlvr.it</a> | Yes    | type             | No    |
|                 |               | text              | tweet's original text                  | I can't believe it was a Virginia Tech student that posted that yik yak today. Just so disappointing ☐  | No     | N/A              | No    |
|                 |               | cleantext         | The text field, minus any commas       | Man, without woman, is nothing => Man without woman is nothing  | No     | N/A              | No    |
|                 |               | from_user         | Twitter username                       | FiremanDave32   | Yes    | screen_name_s    | Yes   |
|                 |               | from_user_id      | user's unique identifier               | 385665827   | No     | N/A              | No    |
|                 |               | time              | Date   created-time (UNIX time)        | 1428951621  | Yes    | created_time_dt  | Yes   |
|                 |               | created_at        | created-time (readable)                | Mon Apr 13 19:00:21 +0000 2015  | No     | N/A              | No    |
|                 |               | iso_language_code | tweet's main language                  | en  | No     | N/A              | No    |
|                 |               | geo_type          | point / polygon                        | point   | No     | N/A              | No    |
|                 |               | geo_coordinates_0 | latitude                               | 43.02099179   | No     | N/A              | No    |
|                 |               | geo_coordinates_1 | longitude                              | -80.44612986  | No     | N/A              | No    |
|                 |               | url               | original URL in tweet                  | <a href="http://twittercounter.com">http://twittercounter.com</a> >The Visitor Widget</a>   | No     | N/A              | No    |
|                 |               | to_user_id        | unique identifier of the reply-to user | 0   | No     | N/A              | No    |
|                 |               | profile_image_url | image URL from the user profile        | http://a0.twimg.com/profile_images/3149217853/0026816c03013356b569a8775af351fb_normal.jpeg  | No     | N/A              | No    |

Figure 22: “tweet” column family

| Table           | Column Family | Column               | Description  | Example   | Stored | Indexed           | Facet |
|-----------------|---------------|----------------------|--|---|--------|-------------------|-------|
| ideal-cs5604f16 | clean-tweet   | clean-text-solr      | 1. bad words censored  | [clean text for Solr and FE]  | Yes    | text_txt          | No    |
|                 |               | clean-text-cla       | 1. bad words censored<br>3. all URLs removed<br>4. stop words removed<br>5. text lemmatized<br>6. remove # or @ symbol from mentions or hashtags | [clean text for CLA]  | No     | N/A               | No    |
|                 |               | clean-text-cta       | 1. bad words censored<br>3. all URLs removed<br>4. stop words removed<br>5. text lemmatized<br>6. remove @ symbol only from mentions, but keep # | [clean text for CTA]  | No     | N/A               | No    |
|                 |               | lemmatized           | text, lemmatized using SNER  | [lemmatized text]   | No     | N/A               | No    |
|                 |               | readable-lang        | Field that stores the language of the tweet, expanded from the abbreviation Twitter provides   | English, Arabic   | No     | language_s        | Yes   |
|                 |               | readable-collection  | Field that stores the name of the collection, found using its id   | #Egypt, heart attack, diabetes  | Yes    | collection_name_s | Yes   |
|                 |               | rt                   | tag for the retweets   | 0 / 1   | No     | N/A               | No    |
|                 |               | geo-location         | readable location from Google API  | Blacksburg,Virginia   | Yes    | location          | Yes   |
|                 |               | created-year         | year extracted from the created-time   | 2015  | No     | t_year_i          | Yes   |
|                 |               | created-month        | month extracted from the created-time  | 11  | No     | t_month_i         | Yes   |
|                 |               | hashtags             | tweet's hashtags   | #hurricane  | Yes    | hashtags          | Yes   |
|                 |               | mentions             | tweet's mentions   | @VT   | Yes    | mentions          | Yes   |
|                 |               | long-url             | extended URL   | <a href="http://www.roanoke.com/news/arrest-made-in-threatening-virginia-tech-yik-yak-post/article_4743fe59-023c-5b26-bebd-662594f7d6ca.html">http://www.roanoke.com/news/arrest-made-in-threatening-virginia-tech-yik-yak-post/article_4743fe59-023c-5b26-bebd-662594f7d6ca.html</a> (from <a href="http://t.co/KEe6gpOMoT">http://t.co/KEe6gpOMoT</a> ) | No     | N/A               | No    |
|                 |               | classification-label | classification label of each tweet   | hurricane   | Yes    | classification_s  | Yes   |
|                 |               | real-world-events    | list of the real world events  | Hurricane Sandy; Hurricane Arthur   | Yes    | events            | Yes   |
|                 |               | snr-people           | extract names from each tweet  | Obama; Jimmy  | Yes    | snr_people        | Yes   |
|                 |               | snr-organizations    | extract organizations from each tweet  | Virginia Tech   | Yes    | snr_org           | Yes   |
|                 |               | snr-locations        | extract locations from each tweet  | New York; London  | Yes    | snr_loc           | Yes   |
|                 |               | tweet-importance     | The importance value for each tweet  | Value between 0 and 1   | No     | t_importance_f    | No    |
|                 |               | user-importance      | The importance value for each user   | Value between 0 and 1   | No     | u_importance_f    | No    |

Figure 23: “clean-tweet” column family

| Table           | Column Family | Column               | Description  | Example   | Stored | Indexed           | Facet |
|-----------------|---------------|----------------------|--|---|--------|-------------------|-------|
| ideal-cs5604f16 | webpage       | 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   |
|                 |               | fetchd-timestamp     | fetchd time (readable)   | Mon Apr 13 19:00:21 +0000 2015                                | No     | fetchd_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    |

Figure 24: “webpage” column family

| Table           | Column Family | Column   | Description          | Example         | Stored | Indexed    | Facet |
|-----------------|---------------|----------|----------------------|-----------------|--------|------------|-------|
| ideal-cs5604f16 | doc-type      | doc-type | type of the document | tweet / webpage | Yes    | doc_type_s | Yes   |

Figure 25: “doc-type” column family

| Table           | Column Family | Column           | Description   | Example  | Stored | Indexed           | Facet |
|-----------------|---------------|------------------|---|--|--------|-------------------|-------|
| Ideal-cs5604f16 | tweet-topic   | label-list       | the top two labels generated by the LDA model for each topic                | Signed,students; event,excited; today,register; april,thanks; community,little | Yes    | topic_label       | No    |
|                 |               | probability-list | each value presents the probability of the tweet belongs to a certain topic | 0.29112; 0.01820; 0.12435; 0.02572; 0.54058                                    | No     | topic_probability | No    |

Figure 26: “tweet-topic” column family



| Table           | Column Family | Column          | Description                               | Example       | Stored | Indexed             | Facet |
|-----------------|---------------|-----------------|---|---------------|--------|---------------------|-------|
| Ideal-cs5604f16 | tweet-cluster | cluster-label   | label of the tweet's cluster              | NAACP stories | Yes    | cluster_label       | No    |
|                 |               | doc-probability | the probability of the doc in the cluster | 0.55167194    | No     | cluster_probability | No    |

Figure 27: “tweet-cluster” column family

### 5.3.3 Social network

The main goal of this task was to develop a query-independent importance methodology for the system’s tweets and web pages based on a graph of these entities that we create. This social network includes interactions between users, tweets, and URLs, with each of the latter serving as nodes in our graph.

There are four subsets in the workflow for this task, which are shown in Figure 28.

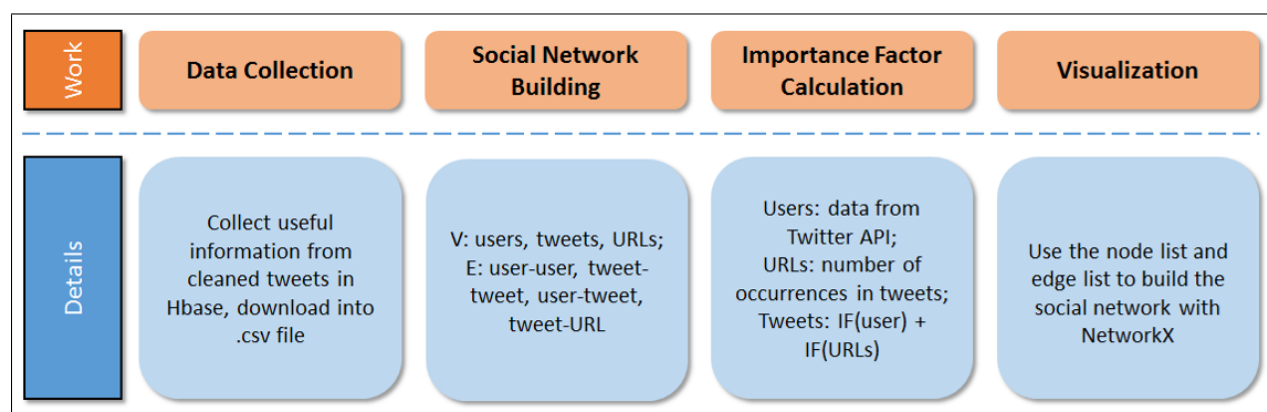


Figure 28: Workflow for building social network

#### 1. Data collection

The first step is the collection of data. We use CSV files for the inputs, so we need to extract cleaned tweets from HBase into a CSV file. Sunshin provided us a Pig script for downloading files to CSV from HBase and we made some modifications based on that to extract files that contains the columns we need for building the social network. The format of the tweet we downloaded from HBase is shown in Figure 29.

| Column name | Collection | Tweet_ID | User_name | From_user_id | mention | RT | URL | tweet |
|-------------|------------|----------|-----------|--------------|---------|----|-----|-------|
|-------------|------------|----------|-----------|--------------|---------|----|-----|-------|

Figure 29: The format of the tweets we collected from HBase

In Figure 30, we also provided some example tweets we extracted from HBase based on the format we defined in Figure 29.

| 1  | Collection           | tweet_id           | user_name       | from_user_id | mention                     | RT | url                   | tweet  |
|----|----------------------|--------------------|-----------------|--------------|-----------------------------|----|-----------------------|--|
| 2  | 3-252190056314441729 | 252190056314441000 | kq4q            | 23021654     |                             | 0  |                       | Before things get better in #Blacksburg there will need to be some major changes within #Hokies foot     |
| 3  | 3-252190544904728576 | 252190544904728000 | rogersdave      | 116772049    | kq4q;kq4q                   | 1  |                       | RT @kq4q: RT @kq4q: Before things get better in #Blacksburg there will need to be some major chang       |
| 4  | 3-252191218061160448 | 252191218061160000 | KatieC_Gardnerr | 620369524    |                             | 0  |                       | I think that's having what it takes. No one can be perfect all the time. Love my school, love my Hokies, |
| 5  | 3-252192541770924035 | 252192541770924000 | tyler_price5    | 467052116    |                             | 0  | http://t.co/jhEckhwj  | So seductive #BrunoTheBear #BlacksBurg http://t.co/jhEckhwj  |
| 6  | 3-252192641167540225 | 252192641167540000 | cadencroy       | 339920627    | tyler_price5;tyler_price5   | 1  | http://t.co/jhEckhwj  | RT @tyler_price5: RT @tyler_price5: So seductive #BrunoTheBear #BlacksBurg http://t.co/jhEckhwj          |
| 7  | 3-252194298995560449 | 252194298995560000 | Next3Days       | 37722313     |                             | 0  | http://t.co/Pun7VfK   | Charleston, SC's Megan Jean & the KFB play @ 622 North #Blacksburg - 10pm - Blend of gypsy, ci           |
| 8  | 3-252198581623197697 | 252198581623197000 | jacobhilliard   | 475504587    | tyler_price5;tyler_price5   | 1  | http://t.co/jhEckhwj  | RT @tyler_price5: RT @tyler_price5: So seductive #BrunoTheBear #BlacksBurg http://t.co/jhEckhwj          |
| 9  | 3-252210978614292480 | 252210978614292000 | handshake20     | 16891682     | CITGAPFunds                 | 0  | http://t.co/jaUxYggu  | GAP 50 Entrepreneur Award Finalists @CITGAPFunds in #Roanoke #Blacksburg: http://t.co/jaUxYggu           |
| 10 | 3-252211742346735616 | 252211742346735000 | handshake20     | 16891682     | TheTechCouncil;CITGAPFun    | 0  | http://t.co/jaUxYggu  | Members of @TheTechCouncil among @CITGAPFunds GAP 50 Entrepreneur Award Finalists #Roanoke               |
| 11 | 3-252214889223974912 | 252214889223974000 | handshake20     | 16891682     | CITGAPFunds                 | 0  | http://t.co/jaUxYggu  | 39 Finalists in #Roanoke #Blacksburg alone. Thanks @CITGAPFunds for #GAP50 Entrepreneur Awards           |
| 12 | 3-252217683985199104 | 252217683985199000 | TheTechCouncil  | 172744497    | handshake20;handshake20;    | 1  | http://t.co/jaU       | RT @handshake20: RT @handshake20: Members of @TheTechCouncil among @CITGAPFunds GAP 50                   |
| 13 | 3-252221633052426240 | 252221633052426000 | j333sh          | 181657880    |                             | 0  | http://t.co/kPX87uPy  | Biggest slug ever. #blacksburg http://t.co/kPX87uPy  |
| 14 | 3-252226820076867584 | 252226820076867000 | northstarfamily | 46780124     |                             | 0  | http://t.co/kBfBznHW  | Three services tomorrow: 8:30, 10:00, 11:45. We would be honored to have you visit! #dontgobe #bla       |
| 15 | 3-252262995818852353 | 252262995818852000 | maegowin        | 329915153    | jlockwoodvt                 | 0  | http://t.co/L2G7Rv59  | @jlockwoodvt #rhymetimes #wuvt #blacksburg #localradio http://t.co/L2G7Rv59                              |
| 16 | 3-252269034102484992 | 252269034102484000 | tavybick        | 364029904    | northstarfamily;northstarfa | 1  | http://t.             | RT @northstarfamily: RT @northstarfamily: Three services tomorrow: 8:30, 10:00, 11:45. We would b        |
| 17 | 3-252281830533894144 | 252281830533894000 | DylanCasciano   | 589564268    | meglaflan                   | 0  |                       | Welcome to VT @meglaflan #blacksburg #pterodactyl  |
| 18 | 3-252281924117213184 | 252281924117213000 | meglaflan       | 482507340    | DylanCasciano;DylanCasciar  | 1  |                       | RT @DylanCasciano: RT @DylanCasciano: Welcome to VT @meglaflan #blacksburg #pterodactyl                  |
| 19 | 3-252284697059672065 | 252284697059672000 | Timw182         | 93790460     | DylanCasciano;DylanCasciar  | 1  |                       | RT @DylanCasciano: RT @DylanCasciano: Welcome to VT @meglaflan #blacksburg #pterodactyl                  |
| 20 | 3-252361534930579457 | 252361534930579000 | BlacksburgEye   | 550607368    |                             | 0  | http://t.co/XoXdrCzA2 | #Blacksburg Sign up and come see your Blacksburg Eye team run/walk the ColorMeRad 5K. Sunday, O          |
| 21 | 3-252392980097413120 | 252392980097413000 | BlacksburgWx    | 829590415    |                             | 0  |                       | *quietly shines some sun on saddened #Blacksburg; staying silent; thinking about that awful game* #      |
| 22 | 3-252393184724934656 | 252393184724934000 | BlacksburgWx    | 829590415    | LibertyPowers;LibertyPowe   | 1  | http://t.co/qNgYxvc1  | RT @LibertyPowers: RT @LibertyPowers: More signs of fall in #Blacksburg. #nrv #weather http://t.co/      |
| 23 | 3-252402752746037248 | 252402752746037000 | andreabadgley   | 82769136     |                             | 0  | http://t.co/26c1s9D   | taking the fam to the Sinkland Farms pumpkin festival today #woohoo! I love fall. http://t.co/26c1s9D    |

Figure 30: Examples of the tweets we collected from HBase

**2. Creating the social network** For the social network, we needed to construct a graph  $G(V, E)$ , where  $V$  is the set of nodes and  $E$  is the set of edges. We first needed to define the nodes and edges based on our demand. Since we wanted to include as much information as we can in the social network, we decided to include the following three kinds of nodes in the node set  $V$  for the social network.

**(a) Users:**

Users are one of the most important factors in the social network, serving as active entities that have interactions with one another via tweets, retweets, mentions, following, and more.

**(b) Tweets:**

Tweets are the primary source of entity-entity interaction in our social network, serving to connect URLs and users. Tweets can also be important in and of themselves, with interactions forming between tweets.

**(c) URLs:**

The URLs also include some useful information (such as source quality) that we could use to help rank tweets in queries.

For the edge set  $E$ , we need to consider all possible interactions between and among those three kinds of nodes. After some research, we found that there are basically four categories of interactions between these nodes. We listed them below and explained how we obtain them.

**(a) User-to-user edges:**

These edges are undirected edges between users, and are based on Twitter retweets (RT) and mentions (@). If we find a tweet that mentions another user in it, we will add an edge between the user who wrote the tweet and the user who is mentioned in that tweet. Additionally, if we find a tweet that retweets another, we will add an edge between the user who retweeted the tweet and the user who wrote the original tweet.

**(b) Tweet-to-tweet edges:**

These edges are undirected edges between tweets, and are based on Twitter retweets (RT). If we find a tweet that retweets another, we will add an edge between those two tweets.

(c) **User-to-tweet edges:**

These edges connect users to tweets and are undirected. If a user writes a tweet, then there is an edge between this user and the tweet he wrote.

(d) **Tweet-to-URL edges:**

These edges connect tweets and URLs and are undirected. If a tweet includes an URL, then there is an edge between this tweet and the URL it includes.

Figure 31 shows the structure of the social network as we defined above. The blue circles represent URLs, the green circles represent tweets and the red circles represent users. We decided to use undirected edges to represent the interactions between nodes for simplicity of calculation. That said, a strong case could be made for the use of directed edges, especially in the case of retweet relations.

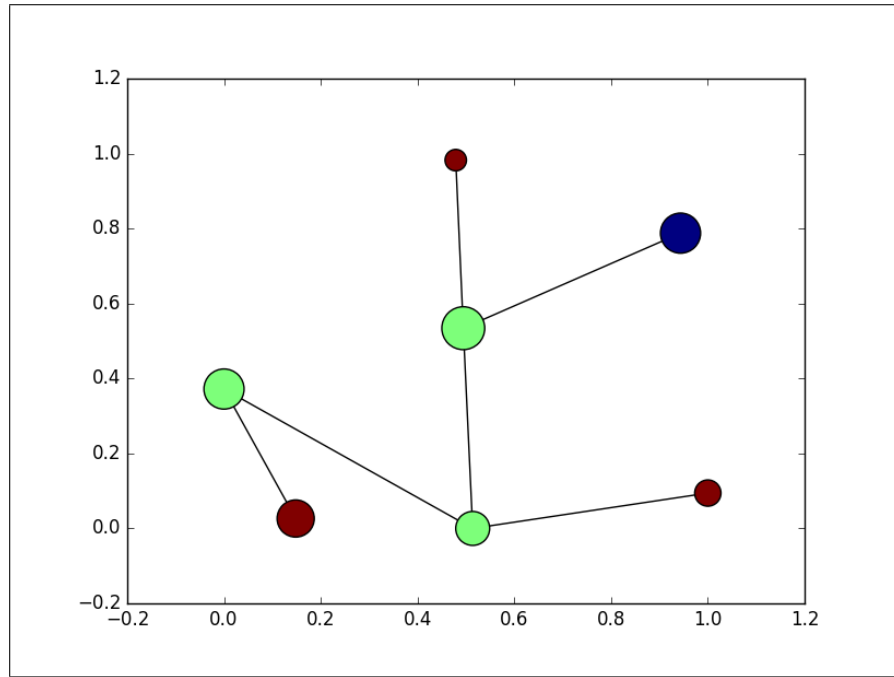


Figure 31: Structure of the social network

**3. Calculate importance factor (IF)** The intuition behind creating the social network is to help improve the performance of queries. Therefore, after building up the whole structure of the social network, the next job is to derive appropriate algorithms to assign weights to each node in the network, reflecting that node's relevant importance. We have no information about the query, so the importance value we compute will only use the social network graph.

(a) **Calculate users importance factors**

In order to rank users, one must first identify the features that they wish to rank on. According to the user information that we can extract using the Twitter API, we decided to use the following five categories of features to rank users:

**i. Followers count ( $c(\text{followers})$ ):**

Indicates the number of followers of users.

**ii. Friends count ( $c(\text{friends})$ ):**

Indicates the number of friends of users.

**iii. Statuses count ( $c(\text{statuses})$ ):**

Indicates the number of tweets (including retweets) issued by users.

**iv. Favorites count ( $c(\text{favorites})$ ):**

Represents the total number of favorite clicks received by users.

**v. Listed count ( $c(\text{listed})$ ):**

Represents the number of public lists that this user is a member of.

The user importance factor is calculated by the summation of all the attributes multiplied by the weights as shown in Equation 1.

$$\mathbf{IF}(\text{user}) = w_1 \times c(\text{followers}) + w_2 \times c(\text{friends}) + w_3 \times c(\text{statuses}) + w_4 \times c(\text{favorites}) + w_5 \times c(\text{listed}) \quad (1)$$

With this Equation 1 in hand, we need to assign weights to each feature. Since what we did is task-independent, it's hard to define an objective to optimize the weights. Therefore, we assign the weights manually based on our knowledge of the importance of each feature we used, reflecting the attributes that we feel should have the greatest importance.

We think that  $c(\text{followers})$ ,  $c(\text{friends})$  and  $c(\text{favorites})$  are more important than the other two features, because if you received a lot of favorites, or you have many friends, or you have many followers, it means that you are connected to more people than others, which makes you more likely to play an important role in the social network.

What's more, we assign a lower weight to the statuses count (which keeps track of the number of tweets a user issues) than the list count (which is the number of lists that others have put a user on) because creating a large volume of tweets is not as important as the user-vetting mechanism of being added to a list (which confers a sense of authority on that user). Therefore, the final equation we used to calculate the important factors for users is shown in Equation 2.

$$\mathbf{IF}(\text{user}) = 0.25 \times c(\text{followers}) + 0.25 \times c(\text{friends}) + 0.1 \times c(\text{statuses}) + 0.25 \times c(\text{favorites}) + 0.15 \times c(\text{listed}) \quad (2)$$

We scale the importance factor to  $[0, 1]$  for intuitive comparison of nodes in the network.

**(b) Calculate URLs importance factors**

For ranking URLs, we just simply count how many times each specific  $\text{URL}_i$  is mentioned in all of the tweets in the whole collection  $c(\text{URL}_i)$ . In order to compare these nodes to the other nodes in the social network, we normalize the importance value by dividing it

by the sum of the count of all URLs  $\sum_k c(\text{URL}_k)$ . The equation we used to define the importance factor of URLs is shown in Equation 3.

$$\mathbf{IF}(\text{URL}_i) = \frac{c(\text{URL}_i)}{\sum_k c(\text{URL}_k)} \quad (3)$$

**(c) Calculating tweets importance factors**

In order to rank tweets, we decided to take advantage of the social network and use the user and the URL connected to the tweet to compute the importance factor of tweets. Considering the following two facts: 1) if the user who posts the tweet has high importance factor, the tweet he posts will also tend to have a high importance factor; 2) If the URL included in a tweet has a high importance factor, then the tweet will also tend to have a high importance factor. Therefore, the tweet importance factor is calculated by the summation of those two importance factors multiplied by the weights as shown in Equation 4.

$$\mathbf{IF}(\text{tweet}) = w_1 \times \mathbf{IF}(\text{user}) + w_2 \times \mathbf{IF}(\text{URL}) \quad (4)$$

Finally, having derived Equation 4, we need to assign weights to the importance factor of the user who posted the tweet and the importance factor of the URL that is mentioned in the tweet ( $\mathbf{IF}(\text{URL}) = 0$  if the tweet does not include any URL). We presume that the importance factor of a user should be weighted more highly than the importance factor of URLs, so we assign the weights shown in Equation 5 to calculate the important factors for tweets.

$$\mathbf{IF}(\text{tweet}) = 0.7 \times \mathbf{IF}(\text{user}) + 0.3 \times \mathbf{IF}(\text{URL}) \quad (5)$$

**(d) Importance factor summary**

To conclude our overview of the importance factor calculations, we now point out a few issues with the methods we have discussed, and areas where they could be improved. To begin, each of the equations used to derive an importance value is somewhat simplistic and takes into account only a node's direct neighbors, as opposed to nodes in neighborhoods many steps away, which PageRank and similar algorithms consider [56]. This was done for simplicity, but our method might not capture as much information about the network as a result. An additional point of concern would be the comparability of the different importance values for tweets, URLs, and users, and the interpretation of each such value. The normalization of the values for users and URLs means that the vast majority of each kind of entity will have very small values. Specifically for URLs, the importance factor will have an inverse relationship with the size of the collection we perform the analysis on, as intuitively, many URLs are likely to be unique, and even those that get repeated will be a small proportion of the total URLs shared. Hence, it might not make sense to add this raw value to user importance to derive the importance value for a tweet. Rather, some sort of adjustment to the URL value might be required before doing so. Finally, an argument could be made that a tweet's importance factor should not include consideration of URLs it mentions, as the more popular a URL is, the more tweets mention it, and the less important any single one of them becomes to the URL's network connectivity.

#### 4. Visualization

Since we were responsible for constructing the social network, visualization of the network was a natural extension of our work. To that end, we did some research into visualization tools that our team or the front-end team can use to visualize sub-graphs. We decided to use NetworkX, because we were familiar with NetworkX and it was easier for us to use it to visualize the results.

NetworkX is a Python language software package for the creation, manipulation, and study of the structure, dynamics, and function of complex networks. With NetworkX you can load and store networks in standard and nonstandard data formats, generate many types of random and classic networks, analyze network structure, build network models, design new network algorithms, draw networks, and much more. Since we were using Python to create the social network, it was easy to use NetworkX to do some simple visualization of sub-graphs [63].

Finally, we used NetworkX to visualize the social network we built and the results are shown in Figure 32. In Figure 32, the green circles represent tweets, the red circles represent users and the blue circles represent URLs. We use the importance factor of each node to decide the size of that node, so the bigger the node is, the more important it is.

To construct Figure 32, we used 300 tweets from Collection z\_3. In the graph, there are:

- 300 tweet nodes
- 158 user nodes
- 110 URL nodes

Additionally, there are:

- 73 user-user edges
- 54 tweet-tweet edges
- 300 user-tweet edges
- 140 tweet-URL edges

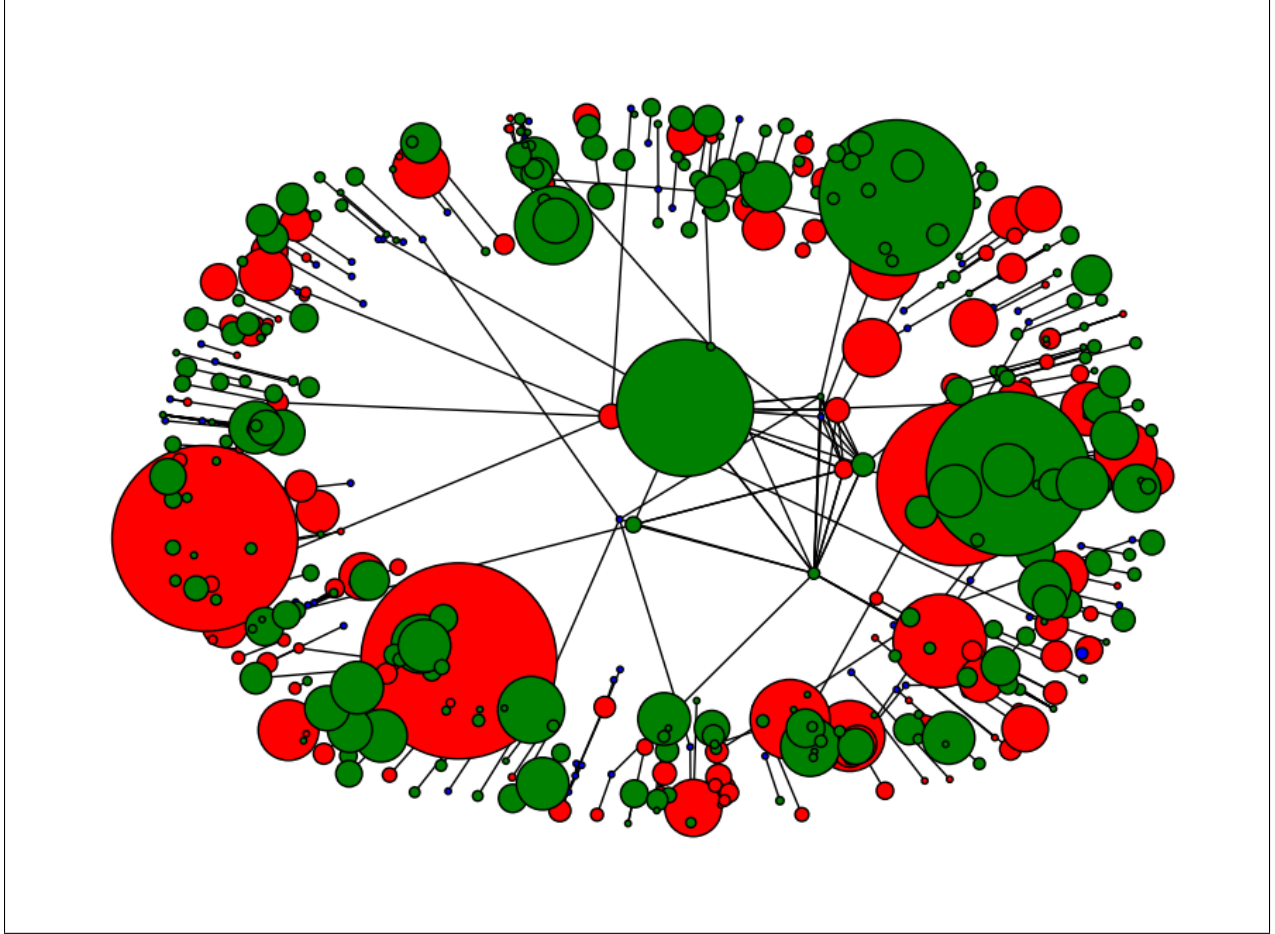


Figure 32: Visualization of the social network using NetworkX

In Figure 32, it is hard to see the blue circles and the reason is that in this tweet collection, there are only a few URLs that are common across tweets. Therefore, based on our algorithm, the importance value for most of the URLs are very small, which makes the blue nodes hard to see. Figure 33 shows a zoomed in version of Figure 32 to make it possible to see the URL nodes.

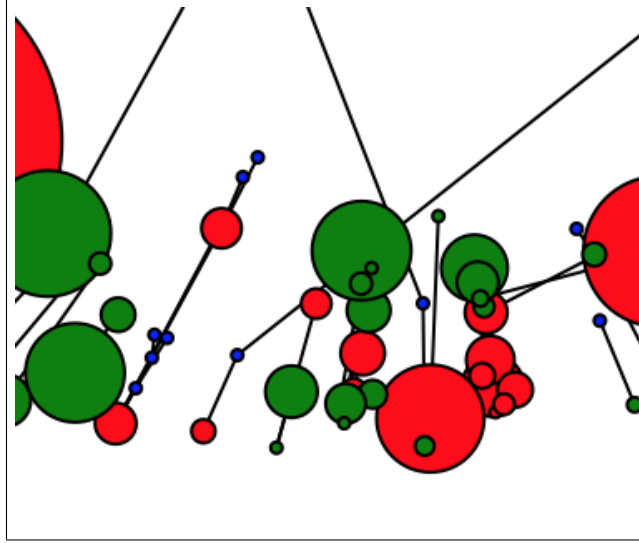


Figure 33: Enlarged bottom part of the graph to see the blue nodes which represent URLs



## 6 User manual

We present a user’s manual for those who might use the project results, access our data sets, or leverage the processing our team is studying are provided in this section.

### 6.1 Incremental Update from MySQL to HDFS

#### 6.1.1 Transfer SQL data to the MySQL database

We assume here that you have access to the test MySQL data that was given to us by Sunshin. We have provided a copy of this that can be found in [64].

Our scripts contain some hard-coded values for base paths of the files that we save to. This would need to be changed per user system design and requirements.

If you are looking to run these scripts locally first instead of on a cluster, that should be okay. Just make sure that you have MySQL installed on your machine (see section 7.2.2 for installation instructions), and download our script from GitHub.

Use [“insert\\_sample\\_data\\_in\\_MySQL.bash”](#) to transfer the test SQL data to a MySQL database. To run this script, follow what is shown in Figure 34

```

[ubuntu@ubuntu: InfoStorage_Project$ ./insert_sample_data_in_MySQL.bash
[ubuntu@ubuntu: InfoStorage_Project$ ./mysql_login.bash
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 61
Server version: 5.5.52-0ubuntu0.14.04.1 (Ubuntu)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show tables;
+-----+
| Tables_in_test_faiz |
+-----+
| Archives              |
| z_312                  |
+-----+
2 rows in set (0.00 sec)

mysql> select count(*) from z_312;
+-----+
| count(*) |
+-----+
|    155657 |
+-----+
1 row in set (0.00 sec)

```

Figure 34: Transfer test SQL data into MySQL table

### 6.1.2 Use pt-archiver to transfer tweets to the ArchiveDB, and save to a file

We assume that you are working on a computer that has pt-archiver installed on it (see section 7.2.13 for the install instructions).

Use [“transfer\\_data\\_using\\_pt-archiver.bash”](#) to transfer tweets from a given table to the ArchivesDB table. This script also simultaneously creates a text file for the tweets. We use this file to do the incremental update of tweets from the MySQL server to the HDFS. See Figure 35 on how to run this script.

```

[ubuntu@ubuntu: InfoStorage_Project$ ./transfer_data_using_pt-archiver.bash
[sudo] password for ubuntu:
Started at 2016-11-22T00:43:36, ended at 2016-11-22T00:45:09
Source: D=test_faiz,h=localhost,p=...,t=z_312,u=root
Dest:   D=test_faiz,h=localhost,p=...,t=Archives,u=root
SELECT 155657
INSERT 155657
DELETE 155657

```

| Action     | Count  | Time    | Pct   |
|------------|--------|---------|-------|
| inserting  | 155657 | 29.5238 | 31.50 |
| deleting   | 155657 | 24.9840 | 26.66 |
| select     | 155658 | 16.6897 | 17.81 |
| commit     | 311316 | 7.1376  | 7.62  |
| print_file | 155657 | 0.6792  | 0.72  |
| other      | 0      | 14.7147 | 15.70 |

```

Successfully transferred tweets to the ArchiveDB, and saved as a file.

```

Figure 35: Transfer tweets to the ArchiveDB, and a text file

Figure 36 shows that all the tweets in the original table get deleted when they are moved to the ArchiveDB. Pt-archiver locks the table it works on. This makes sure that the concerned table does not get dynamically updated while pt-archiver works on it. The text file created for the tweets is used for the incremental update to HDFS.

```

ubuntu@ubuntu: InfoStorage_Project$ ./mysql_login.bash
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 68
Server version: 5.5.52-0ubuntu0.14.04.1 (Ubuntu)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> select count(*) from z_312;
+-----+
| count(*) |
+-----+
|         0 |
+-----+
1 row in set (0.00 sec)

mysql> select count(*) from Archives;
+-----+
| count(*) |
+-----+
|    155657 |
+-----+
1 row in set (0.00 sec)

mysql> Bye
ubuntu@ubuntu: InfoStorage_Project$ ls ~/sample_msql_data/
2016-11-22-test_faiz.z_312  Dump20160919.sql

```

Figure 36: All tweets deleted from the original table, moved to the ArchiveDB, and a separate file

### 6.1.3 Cleaning the text file of tweets

You need to run a single bash script to remove unnecessary newline characters, and insert newline characters at the end of each line. We also removed single quotes, and double quotes from every line since their presence was making it difficult to convert CSV files into Avro files.

Use [“cleanup\\_tweets.bash”](#) to perform cleaning of the text file of the tweets. See Figure 37 on how to run this script.

```

faiz89@viz1: csv_data$ ~/git/InfoStorage_Project/cleanup_tweets.bash 2016-11-22-test_faiz.z_312
faiz89@viz1: csv_data$ ls -l
total 0
-rw-rw-r-- 1 faiz89 faiz89 0 Nov 25 16:19 2016-11-22-test_faiz.z_312.csv

```

Figure 37: Clean the tweets files

The argument you provide to this script is the file generated by pt-archiver as discussed in Section 6.1.2.

#### 6.1.4 Converting the CSV file into an Avro file and copying it to HDFS

The cleaned CSV file obtained in Section 6.1.3 first needs to be converted into an Avro file format. We assume here that csv2avro[31] is already installed on the system.

Based on the name of the file, the Avro files need to be put at specific locations on HDFS. For example, a part of the file name shown in Figure 39 is “z\_312”. We need this part to understand where to put this file on the HDFS file system. Figure 38 shows the different locations where an Avro file can be put on the HDFS system in our system.

```

cs5604f16_cmt@node1: ~$ hadoop fs -ls /collections/tweets-705/
Found 662 items
drwxr-xr-x - hdfs supergroup      0 2015-09-30 15:47 /collections/tweets-705/archives
drwxr-xr-x - hdfs supergroup      0 2015-09-30 15:48 /collections/tweets-705/processes
drwxr-xr-x - hdfs supergroup      0 2015-09-30 15:49 /collections/tweets-705/rawstream
drwxr-xr-x - hdfs supergroup      0 2015-09-30 15:51 /collections/tweets-705/z_1
drwxr-xr-x - hdfs supergroup      0 2015-09-30 15:51 /collections/tweets-705/z_10
drwxr-xr-x - hdfs supergroup      0 2015-09-30 15:52 /collections/tweets-705/z_101
drwxr-xr-x - hdfs supergroup      0 2015-09-30 15:52 /collections/tweets-705/z_102
drwxr-xr-x - hdfs supergroup      0 2015-09-30 15:53 /collections/tweets-705/z_103
drwxr-xr-x - hdfs supergroup      0 2015-09-30 15:54 /collections/tweets-705/z_104
drwxr-xr-x - hdfs supergroup      0 2015-09-30 15:54 /collections/tweets-705/z_105
drwxr-xr-x - hdfs supergroup      0 2015-09-30 15:54 /collections/tweets-705/z_106
drwxr-xr-x - hdfs supergroup      0 2015-09-30 15:55 /collections/tweets-705/z_107
drwxr-xr-x - hdfs supergroup      0 2015-09-30 15:56 /collections/tweets-705/z_108
drwxr-xr-x - hdfs supergroup      0 2015-09-30 15:56 /collections/tweets-705/z_109
drwxr-xr-x - hdfs supergroup      0 2015-09-30 15:57 /collections/tweets-705/z_11
drwxr-xr-x - hdfs supergroup      0 2015-09-30 15:57 /collections/tweets-705/z_110
drwxr-xr-x - hdfs supergroup      0 2015-09-30 15:57 /collections/tweets-705/z_111
drwxr-xr-x - hdfs supergroup      0 2015-09-30 15:58 /collections/tweets-705/z_112
drwxr-xr-x - hdfs supergroup      0 2015-09-30 15:58 /collections/tweets-705/z_113
drwxr-xr-x - hdfs supergroup      0 2015-09-30 15:59 /collections/tweets-705/z_114
drwxr-xr-x - hdfs supergroup      0 2015-09-30 15:59 /collections/tweets-705/z_115
drwxr-xr-x - hdfs supergroup      0 2015-09-30 16:00 /collections/tweets-705/z_116
drwxr-xr-x - hdfs supergroup      0 2015-09-30 16:00 /collections/tweets-705/z_117
drwxr-xr-x - hdfs supergroup      0 2015-09-30 16:00 /collections/tweets-705/z_118
drwxr-xr-x - hdfs supergroup      0 2015-09-30 16:01 /collections/tweets-705/z_119
drwxr-xr-x - hdfs supergroup      0 2015-09-30 16:01 /collections/tweets-705/z_12
drwxr-xr-x - hdfs supergroup      0 2015-09-30 16:02 /collections/tweets-705/z_120
drwxr-xr-x - hdfs supergroup      0 2015-09-30 16:02 /collections/tweets-705/z_121

```

Figure 38: Different folders on the HDFS system for tweets coming from different tables on the MySQL server

Use [“convert\\_csv\\_avro.bash”](#) as shown in Figure 39 to convert the CSV file into an Avro file. This script also copies the file from the MySQL server to the HDFS server to its correct location.

```
faiz89@viz1: InfoStorage_Project$ ./convert_csv_avro.bash 2016-12-07-infostorage.z_312.csv schema.avsc
/var/lib/gems/2.3.0/gems/csv2avro-1.3.1/lib/csv2avro/converter.rb:17: warning: circular argument referen
ce - schema
{"timestamp":"2016-12-07 08:15:11 UTC","level":"INFO","logger":"csv2avro","event":{"name":"started_conve
rting","filename":"2016-12-07-infostorage.z_312.csv","monitored":true,"title":"Started converting 2016-1
2-07-infostorage.z_312.csv","text":"Started converting 2016-12-07-infostorage.z_312.csv"},"message":"Sta
rted converting 2016-12-07-infostorage.z_312.csv"}
{"timestamp":"2016-12-07 08:15:23 UTC","level":"INFO","logger":"csv2avro","event":{"name":"finished_conv
erting","filename":"2016-12-07-infostorage.z_312.csv","monitored":true,"title":"Finished converting 2016
-12-07-infostorage.z_312.csv","text":"Finished converting 2016-12-07-infostorage.z_312.csv, processed 15
5658 lines in total."},"metrics":[{"name":"lines_processed","value":155658,"type":"counter"}],"message":
"Finished converting 2016-12-07-infostorage.z_312.csv"}
Conversion to avro successful!
Table name is z_312.
We will copy 2016-12-07-infostorage.z_312.avro to /collections/tweets-705/z_312/
cs5604f16_cmt@hadoop.dlib.vt.edu's password:
```

Figure 39: Convert a CSV file into Avro file format

### 6.1.5 Merging the Avro files on HDFS

Sunshin informed us that the block size of the HDFS system on the cluster is 256 megabytes. This means that even if you want to store a 1 megabyte file on the HDFS system, it will occupy 256 megabytes, essentially wasting 255 megabytes.

We needed to come up with a solution that merges the Avro files on the HDFS system to save disk space. Figure 40 shows two Avro files on the HDFS file system. We use Avro Tools [65] to do the same.

```
[cs5604f16_cmt@node1 ~]$ hadoop fs -ls faiz_testing
Found 2 items
-rw-r--r-- 3 cs5604f16_cmt cs5604f16_cmt 57104700 2016-11-29 01:23 faiz_testing/2016-11-22-test_fai
z.z_312.avro
-rw-r--r-- 3 cs5604f16_cmt cs5604f16_cmt 57104700 2016-11-29 01:23 faiz_testing/2016-11-22-test_fai
z.z_312.avro-copy
```

Figure 40: Two Avro files to be merged on HDFS

Use [“merge\\_avro.bash”](#) as shown in Figure 41 to merge the two Avro files into one big Avro file. The files are deleted after being merged.



```

[[cs5604f16_cmt@node1 avro-files]$ ./merge_avro.bash faiz_testing/2016-11-22-test_faiz.z_312.avro fa
iz_testing/2016-11-22-test_faiz.z_312.avro-copy faiz_testing
16/12/07 03:47:39 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 144
0 minutes, Emptier interval = 0 minutes.
Moved: 'hdfs://nameservice1/user/cs5604f16_cmt/faiz_testing/2016-11-22-test_faiz.z_312.avro' to tra
sh at: hdfs://nameservice1/user/cs5604f16_cmt/.Trash/Current
16/12/07 03:47:39 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 144
0 minutes, Emptier interval = 0 minutes.
Moved: 'hdfs://nameservice1/user/cs5604f16_cmt/faiz_testing/2016-11-22-test_faiz.z_312.avro-copy' t
o trash at: hdfs://nameservice1/user/cs5604f16_cmt/.Trash/Current
Merge of faiz_testing/2016-11-22-test_faiz.z_312.avro and faiz_testing/2016-11-22-test_faiz.z_312.a
vro-copy successful!
[[cs5604f16_cmt@node1 avro-files]$ hadoop fs -ls faiz_testing/
Found 1 items
-rw-r--r--   3 cs5604f16_cmt cs5604f16_cmt  114208513 2016-12-07 03:47 faiz_testing/combined.avro
[[cs5604f16_cmt@node1 avro-files]$

```

Figure 41: Merge the two Avro files on the HDFS file system

This completes the incremental update feature from the MySQL server to HDFS.

## 6.2 Incremental update from HDFS to HBase

This section describes how to build and use the scripts provided in our GitHub repository [64] for the incremental update from HDFS to HBase. It is assumed that these scripts are enacted on the temporary Avro file created in Section 6.1, before it is merged.

### 6.2.1 Preparing the pipeline

Our pipeline requires that the Hadoop ecosystem components discussed in the report are installed, as well as Python 2.6.6. Additionally, while most of the pipeline consists of Pig and bash scripts, a component of it (the lemmatization and Stanford named entity recognition step) is implemented as a Java program that must be built. Please see Section 7 for more details on software specifics.

This Java program is managed by Gradle [66], and can be found under the directory */tweet-processing/nlp/* in the code that we have included with this submission. To build the project, run the command “gradle build.” The resulting jar file will be built and saved to */tweet-processing/nlp/build/libs/*. To add this jar to the pipeline, move it to the folder */tweet-processing/pipeline/*.

### 6.2.2 Running our scripts

With the jar file in place, one can now run a simple bash script to process process tweets: the file */tweet-processing/pipeline/process.sh*. This file is configured to work on the IDEAL/GETAR system: instructions on how to accommodate another system are included in Section 7.

The script we have developed takes three arguments: the name of an HBase table, and two collection numbers of the collections to be processed (inclusive). The scripts that *process.sh* calls use these collection numbers to derive the location of an Avro file on our HDFS instance (these locations are hard-coded, and should be changed according to the system design of the user).

```
[cs5604f16_cmt@node1 pipeline]$ ./process.sh cmt-test 1 1
```

Figure 42: Running procces.sh

### 6.3 Build social network

This section describes how to use the script for building a social network from a collection, as well as how to visualize the results.

#### 6.3.1 Collect data

We developed a simple script using the Apache PiggyBank [61] to access our HBase table and save the result as a CSV on HDFS. This script can be found in our files under */tweet-processing/csv-download/*.

In this script, a couple of hard-coded values specific to our system have been used to specify the range of tweets to scan in HBase. An additional hard-coded value has been used to specify the name under which to save the CSV on HDFS. These values need to be modified according to the desires of the users of our code.

This scripts writes to HDFS a folder consisting of a number of files representing parts of the true, complete CSV. Concatenating these files yields the complete CSV for further analysis.

```
[cs5604f16_cmt@node1 csv-download]$ ls
download-to-csv.pig piggybank-0.12.0.jar piggybank-0.16.0.jar
[cs5604f16_cmt@node1 csv-download]$ vi download-to-csv.pig
[cs5604f16_cmt@node1 csv-download]$ vi download-to-csv.pig
[cs5604f16_cmt@node1 csv-download]$ hdfs dfs -get test3.csv
[cs5604f16_cmt@node1 csv-download]$ ls
download-to-csv.pig piggybank-0.12.0.jar piggybank-0.16.0.jar test3.csv
[cs5604f16_cmt@node1 csv-download]$ cd test3.csv/
[cs5604f16_cmt@node1 test3.csv]$ ls
part-m-00000 part-m-00002 part-m-00004 part-m-00006 part-m-00008 part-m-00010 _SUCCESS
part-m-00001 part-m-00003 part-m-00005 part-m-00007 part-m-00009 part-m-00011
[cs5604f16_cmt@node1 test3.csv]$ cat p* > out.csv
[cs5604f16_cmt@node1 test3.csv]$ vi out.csv
[cs5604f16_cmt@node1 test3.csv]$
```

Figure 43: Concatenating part files into a single CSV

#### 6.3.2 Use Twitter API

We developed a script for collecting useful information for users from Twitter using Tweepy [67]. Tweepy is open-source, hosted on GitHub, and enables Python to communicate with the Twitter platform and use its API. Using Tweepy, we can easily extract useful information we need from Twitter. This script can be found in our files under */social-network/* named `twitter_api.py`. The red box in Figure 44 shows the range of information that Tweepy can gather from Twitter.



```

def get_users_info(id_list, select_type):
    user_info = []

    for i in range(0, len(id_list), 100):
        id_sub_list = [item for item in id_list[i:i + 100]]

        user_list = api.lookup_users(user_ids=id_sub_list)

        for user in user_list:
            if select_type == 'ALL':
                user_info.append([user.id_str,
                                user.name,
                                user.screen_name,
                                #user.created_at,
                                #user.description,
                                #user.geo_enabled,
                                #user.location,
                                #user.time_zone,
                                user.url,
                                #user.verified,
                                #user.lang,
                                #user.following,
                                user.followers_count,
                                user.friends_count,
                                user.statuses_count,
                                user.favourites_count,
                                user.listed_count])
            elif select_type == 'NAME':
                user_info.append([user.name])
        time.sleep(20)
    return user_info

```

Figure 44: How to use Tweepy to extract useful information from Twitter

In our code, we have included a script called *tweetsSN.py* for calculating the importance factor of users. One can use it directly by providing a list of `user_ids` as input. The red box in Figure 45 shows us providing such a list, and using it to extract user-based information from Twitter.

```

100 def main():
101     ~~~~~
102     configuration()
103     ~~~~~
104     # get user info from the above user ids
105     user_id_list = read_file_into_list_user('reduced.csv')
106     ~~~~~
107     user_inf = get_users_info(user_id_list, 'ALL')
108     ~~~~~
109     user_info = calculate_user_importance(user_inf)
110     ~~~~~
111     ~~~~~
112     # main function
113     main()

```

Figure 45: Providing a list of users and gathering data with Tweepy

### 6.3.3 Build social network and compute importance factor

We also use `tweetsSN.py` to build our social network and compute the importance factors of the social network entities. The input of this function should be a CSV file downloaded from HBase which should include basic tweet information including `tweet_id`, `user_name`, `from_user_id`, `tweet`, `url(detailsabove)`. Along with the CSV file, you also need to provide field names to help identify the meaning of each column. Figure 46 demonstrates the use of this file. The field names you should identify are shown by the blue box in Figure 46.

```

78 with open('reduced_withname.csv', 'rU') as csvfile:
79     fieldnames = ['collection', 'id', 'user', 'from user id', 'mention', 'rt', 'url', 'tweet']
80     reader = csv.DictReader(csvfile, fieldnames=fieldnames)
81     for row in reader:
82         tr={}
83         tr['user_id']=row['from_user_id']
84         tr['tweet_id']=row['id']
85         tr['url'] = row['url']
86         tr['tweet'] = row['tweet']
87         tr['user'] = row['user']
88         tweets_record.append(tr)

```

Figure 46: The input of the script for building social network

```

331 with open('nodescsvfile.csv', 'wb') as f:
332     w = csv.DictWriter(f, nodesdic[0].keys())
333     w.writeheader()
334     w.writerows(nodesdic)
335
336 with open('edgescsvfile.csv', 'wb') as f:
337     fieldnames = ['from', 'to']
338     w = csv.DictWriter(f, fieldnames=fieldnames)
339     w.writeheader()
340     for row in edgesdic:
341         w.writerow(row)

```

Figure 47: The output of the script for building social network

This script will output two files: the node list (including importance factors), and the edge list. In order to run this script, you also need to identify the names of the output files. For example, we use “nodescsvfile.csv” for the node list and “edgescsvfile.csv” for the edge list, as shown in the red and blue boxes in Figures 47.

### 6.3.4 Visualization using NetworkX

Once you get the node list and edge list, you can then pass the results to `visualization.py`, which we developed to visualize the results using NetworkX. The inputs of this script are exactly the outputs of the `tweetsSN.py` script. You can change the file name of the node list by modifying the file name (Figure 48). We normalized the importance factor to  $[0, 1]$ ; it is hard to visualize if we use the importance factor directly to set the size of the nodes in NetworkX. Therefore, we multiply the importance factor by a factor of 10,000 to obtain the size for each node.

```

14 with open('nodescsvfile.csv', 'rU') as csvfile:
15     reader = csv.DictReader(csvfile)
16     for row in reader:
17         G.add_node(row['id'])
18         node_sizes.append(10000*float(row['if'])+10)
19         labels.append(row['id'])

```

Figure 48: The input of the script for visualization using NetworkX

Also, you should put the file name of the edge list in the red box shown in Figure 49.

```
30 with open('edgecsvfile.csv', 'rU') as csvfile:
31     reader = csv.DictReader(csvfile)
32     for row in reader:
33         G.add_edge(row['to'], row['from'])
```

Figure 49: The output of the script for visualization using NetworkX

## 7 Developer manual

We present a developer’s manual for those who want to understand our data and code and incorporate changes or make enhancements for their own purposes. We will review the technologies that we use in this project, giving a brief introduction to them at first and then diving more into their details and implementation.

### 7.1 IDEAL/GETAR cluster architecture

Our project is implemented on a 20-node + 1 (Solr node) computing cluster, for 88 CPU cores, 704GB of RAM, and 154.3TB of storage. More specifically:

- The Solr node has 8 Intel Xeon cores, and 64GB of RAM.
- The remaining nodes each have Intel i5 (Haswell) 4-core processors and 32GB of RAM.
- The cluster head node has 6TB of storage.
- The remaining nodes have 3TB + 4TB apiece.
- We maintain an 8.3TB NAS Backup.

We utilize Cloudera Hadoop 5.6.0 (CAP program).

### 7.2 Module overviews

There are many parts of the IDEAL [1] and GETAR [2] projects. For a broad overview of the system architecture, see Figure 1. Below are some parts of the bigger architecture components that concern us as the CMT team.

#### 7.2.1 MySQL database

A relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model. Relational database management systems were introduced in the 1970’s. RDBMSs avoid the navigation model employed by old DBMSs. The relational model has relationships between tables using primary keys, foreign keys and indexes. Thus, the fetching and storing of data become faster than the old navigational model. RDBMSs are widely used by enterprises and developers for storing complex and large amounts of data. The most popular RDBMSs are MS SQL Server, DB2, Oracle and MySQL [68]. Figure 50 shows a relational database model of describing a relation between tweets, users, URLs, and other entities.

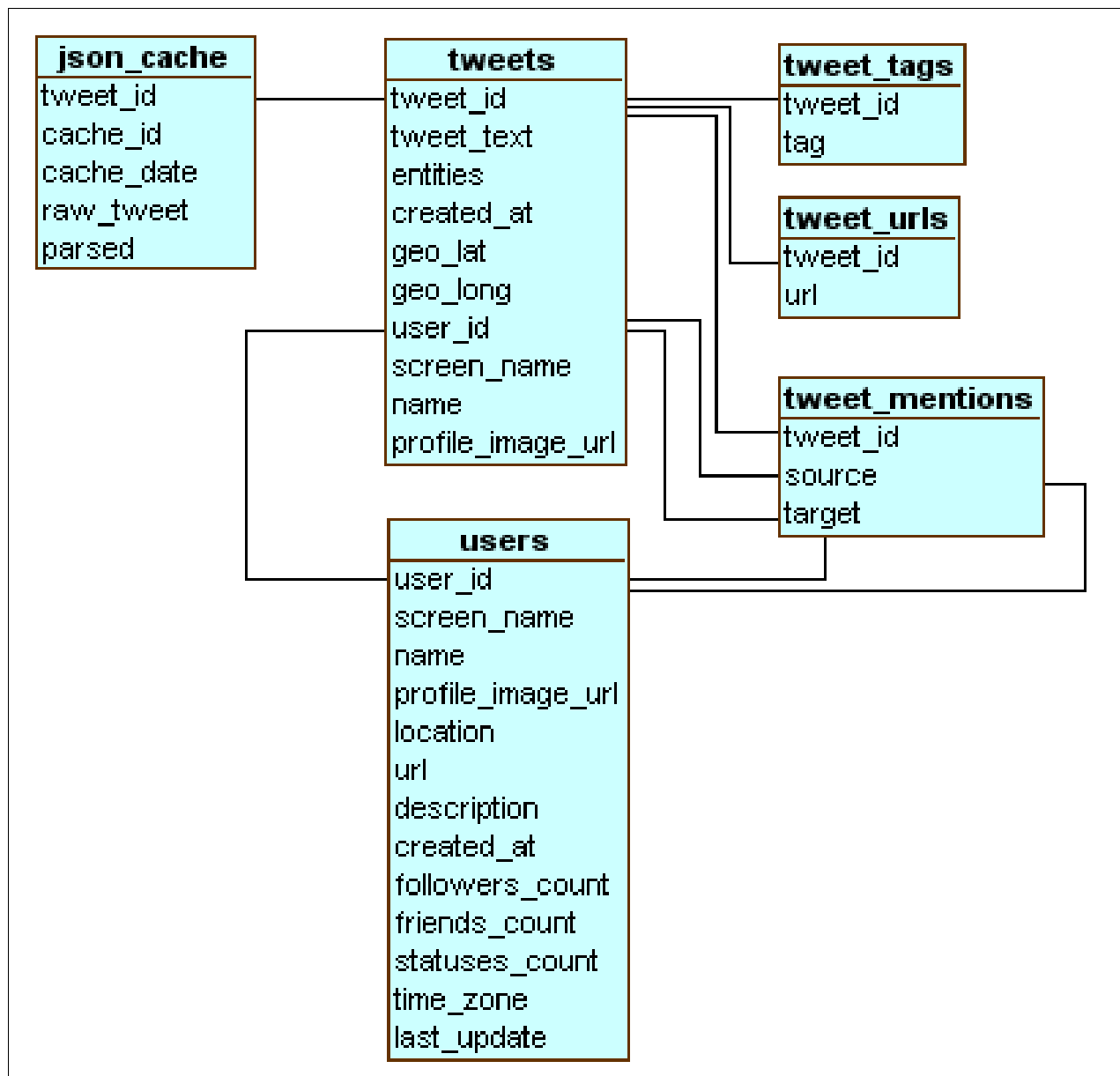


Figure 50: A relational database model describing a relation between tweets, users, URLs, and other entities [17]

MySQL databases consist of any number of tables, made up of rows and columns, that store data. A user that has been given CREATE and DROP permissions on a database can create and remove tables of that database. The CREATE TABLE command simultaneously creates the table and defines its structure (although the structure of the table can later be changed using the ALTER TABLE command).

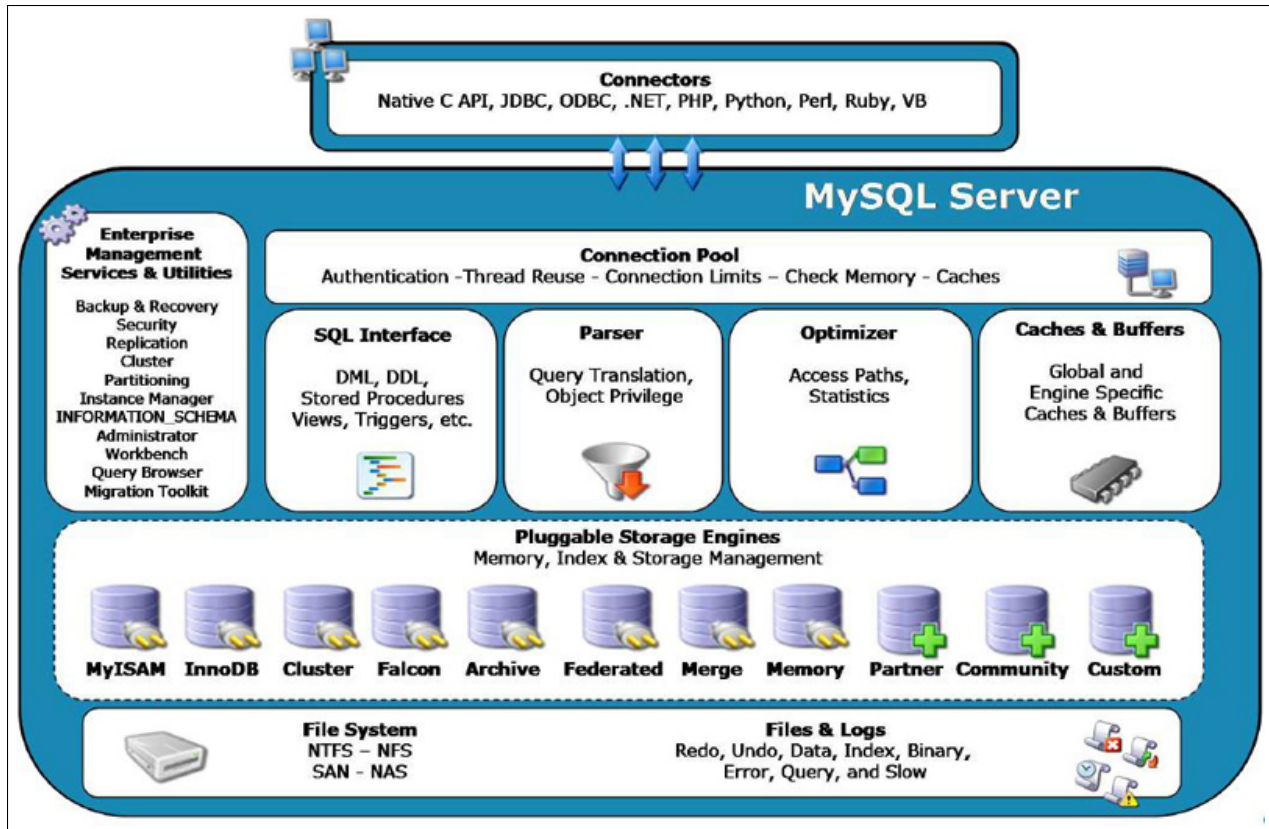


Figure 51: MySQL architecture overview [18]

In this project, we also collect tweets using the YTK [25] tool, which collects tweets directly from Twitter and stores them into a MySQL database.

There are many advantages of using MySQL. Some of them are outlined below [69].

1. Scalability and flexibility - The MySQL database server provides the ultimate in scalability, sporting the capacity to handle deeply embedded applications with a footprint of only 1MB to running massive data warehouses holding terabytes of information [69].
2. High performance - A unique storage-engine architecture allows database professionals to configure the MySQL database server specifically for particular applications, with the end result being amazing performance results [69].
3. Robust transactional support - MySQL offers one of the most powerful transactional database engines on the market. Features include complete ACID (atomic, consistent, isolated, durable) transaction support, unlimited row-level locking, distributed transaction capability, and multi-version transaction support where readers never block writers and vice-versa [69].
4. Strong data protection - Because guarding the data assets of corporations is the number one job of database professionals, MySQL offers exceptional security features that ensure absolute data protection. In terms of database authentication, MySQL provides powerful mechanisms for ensuring only authorized users have entry to the database server, with the ability to block

users down to the client machine level being possible. Finally, backup and recovery utilities provided through MySQL and third party software vendors allow for complete logical and physical backup as well as full and point-in-time recovery [69].

### 7.2.2 MySQL installation & operation

1. **Installation:** Follow the steps on <http://dev.mysql.com/doc/refman/5.7/en/installing.html> to download and install MySQL. Pick the version number for MySQL Community Server you want and the platform you want.
2. **Operation:** Some useful commands for using MySQL are shown in Figure 52.

```
# Access the MySQL shell
mysql -u root -p
# Check what databases are available
SHOW DATABASES;
# Creating a database
CREATE DATABASE database name;
# Delete a MySQL database
DROP DATABASE database name;
# Open up the database we want to use
USE database name;
# Show the tables that the database contains
SHOW tables;
# Create a new MySQL table
CREATE TABLE potluck (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
name VARCHAR(20),
food VARCHAR(30),
confirmed CHAR(1),
signup_date DATE);
# Insert information into each row
INSERT INTO `potluck` (`id`,`name`,`food`,`confirmed`,`signup_date`)
VALUES (NULL, "John", "Casserole","Y", '2012-04-11');
# Update Information in the Table
UPDATE `potluck`
SET
`confirmed` = 'Y'
WHERE `potluck`.`name` ='Sandy';
# Add a Column
ALTER TABLE potluck ADD email VARCHAR(40);
# Add column in a specific spot in the table
ALTER TABLE potluck ADD email VARCHAR(40) AFTER name;
# Delete a Column
ALTER TABLE potluck DROP email;
# Delete rows from the table
DELETE from [table name] where [column name]=[field text];
```

Figure 52: Useful commands for MySQL

There are many good tutorials on learning MySQL. One useful can be found on YouTube [70].



### 7.2.3 Apache Hadoop

Apache Hadoop [71] is a popular tool used for distributed storage and processing. It is an open source software package that is being actively developed by a large community of users, and is designed to efficiently store and process very large data sets on clusters of computers formed from commodity hardware. The heart of Hadoop is called Hadoop Distributed File System (discussed in the next section), but the ecosystem actually consists of several modular components:

1. Hadoop Common - all the libraries and utilities are stored here.
2. HDFS - HDFS is a distributed file system designed to store large collections of data efficiently.
3. Hadoop YARN - this is responsible for managing the compute resources in the cluster. It can be thought of as a job scheduler that manages the compute resources.
4. Hadoop MapReduce - Hadoop's processing backend is called Hadoop MapReduce. As its name suggests this software framework implements the MapReduce [72] programming model developed to process very large data sets. When a particular job is submitted to a Hadoop cluster, the components of the ecosystem divide the necessary work up into small pieces. These small pieces are then distributed in manageable chunks to the cluster's nodes, and the results of each computation are merged together in subsequent aggregation.

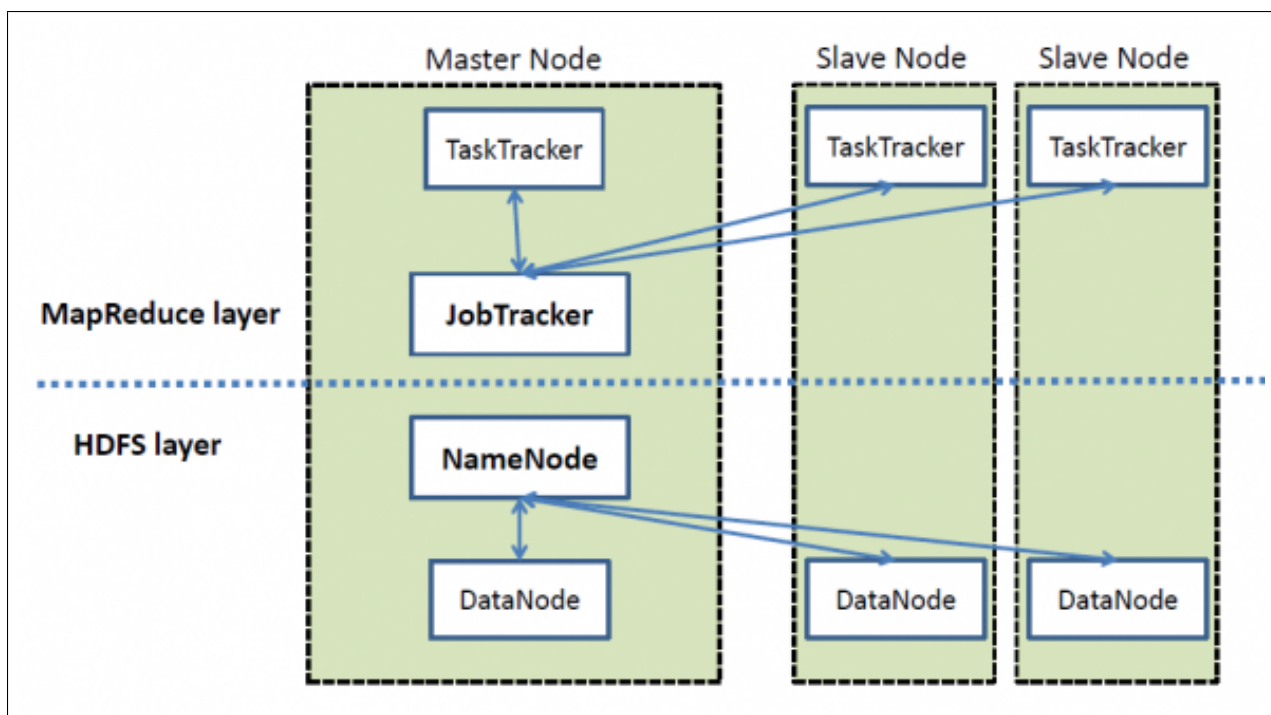


Figure 53: High level architecture of Hadoop [19]

Historically, Hadoop's concept came from the Google File System [73] paper published in 2003. Another paper from Google on MapReduce [74] was published, and it was in 2006 that work on the Hadoop project started.

There are many advantages of using Hadoop. Some of them are outlined below.

1. Scalability - Hadoop can scale to huge amounts of data effortlessly. It can store and distribute big data in a cluster made up of hundreds of nodes working in parallel. Relational databases like MySQL, etc. will not scale like this. This parallel execution of data enables Hadoop to run applications on thousands of nodes involving exabytes of data.
2. Economical to use - scaling up data using relational databases is not cost-effective. Storing large amounts of data in such databases is costly. Hadoop's HDFS is much more economical to use by businesses.
3. Flexible - Hadoop can process structured and unstructured data. This enables businesses to derive useful information from data sources like emails, social media, fraud detection, etc.
4. Fast - since Hadoop distributes the workload onto different nodes in a cluster, the data processing is very fast. This parallel execution is directly proportional to the number of nodes in the cluster. As a fun fact, in 2009, Yahoo! used Hadoop to sort one terabyte of data in 62 seconds [75]!
5. Fault tolerant - this is another important aspect of Hadoop. If at any given point in time, one node fails in the cluster, the master node of Hadoop reassigns its work to some other node. This way, the entire job does not fail, and there is no data lost either.

#### **7.2.4 Apache Hadoop installation & operation**

Hadoop version 5.6.0 is currently being used by the project to process the 1.2 billion tweets.

1. **Installation:**

To install Hadoop on a single node cluster, please follow the instructions mentioned on the Hadoop website [76].

2. **Operation:**

Some useful Hadoop commands that we have been using in our project are given below.

### command for Hadoop

```
## Print the Hadoop version.
hadoop version
## List the contents of the root directory in HDFS.
hadoop fs -ls /
## Report the amount of space used and available on
## currently mounted filesystem
hadoop fs -df hdfs:/
## Count the number of directories, files and bytes under
## the paths that match the specified file pattern
hadoop fs -count hdfs:/
## Run a DFS filesystem checking utility
hadoop fsck - /
## Create a new director named "hadoop" below the
## /user/trainig directory in HDFS. Since you're
## currently logged in with the "training" user ID,
## /user/training is your home directory in HDFS.
hadoop fs -mkdir /user/training/hadoop
## Add a samle text file from the local directory named
## "data" to the new directory you created in HDFS during
## the previous step.
hadoop fs -put data/sample.txt/user/training/hadoop
## List the contents of this new directry in HDFS.
Hadoop fs -ls /user/training/hadoop
```

Figure 54: Useful commands for Hadoop

To learn more about Hadoop, we recommend watching the video posted by HortonWorks [77].

### 7.2.5 HDFS

The Hadoop Distributed File System (HDFS) is a Java-based distributed file system designed for storing large amounts of data reliably across networked machines. As part of the Apache Hadoop framework, it leverages technologies like Apache YARN to act as a redundant and fault-tolerant data-storage solution that can be run on commodity hardware [78].

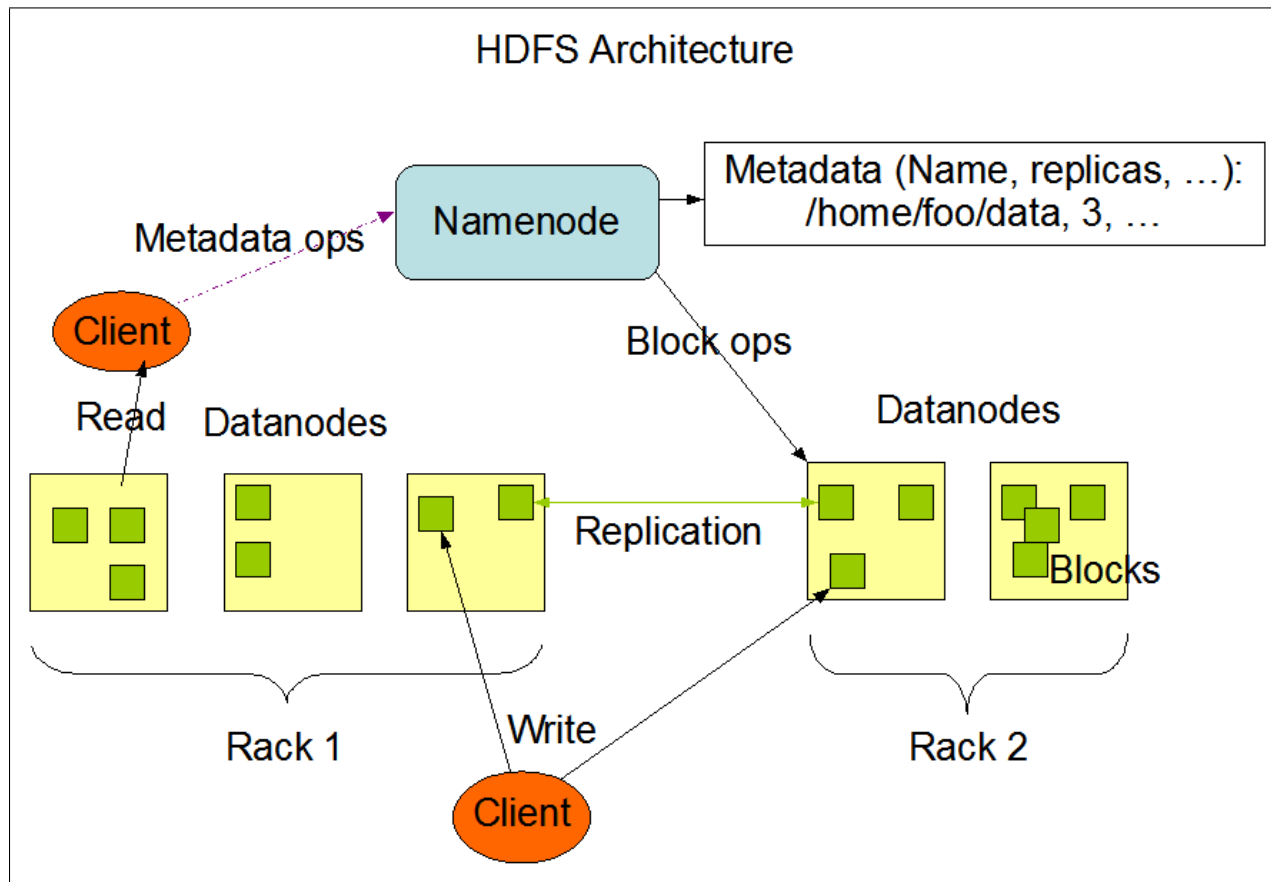


Figure 55: HDFS architecture overview [20]

HDFS implements a traditional hierarchical file system designed around a master-slave architecture. A single *NameNode* serves as a master server that manages the file system name space and serves as an intermediary between clients and the file system itself. Files are stored as a sequence of blocks across slave machines (known as *DataNodes*) and replicated across machines for redundancy. Both block size and replication factor can be set on a per-file basis [79].

HDFS provides a shell interface that lets users interact with the file system in a manner very similar to that of other shells like bash. For example, it supports the traditional *ls* and *mkdir* commands.

Additionally, HDFS provides several API options, including a native Java API and a C wrapper for it, allowing for programmable interaction with the system [79][80].

The IDEAL/GETAR system leverages HDFS to store tweets on the order of billions. Apache Sqoop is used to pull those tweets from an external MySQL database into HDFS in the Avro file format, an efficient format for data serialization [60]. Additionally, an HBase database sits on top of HBase, utilizing its capabilities to store cleaned tweets and webpages for indexing and other analysis. Through the use of Apache Pig, the data in the Avro files can be loaded into this larger database.

### 7.2.6 HDFS installation & operation

1. **Installation:** For the installation of HDFS, please refer to the instructions for the installation of Hadoop. When you install Hadoop, it will install HDFS.
2. **Operation:** Some useful commands for using HDFS are shown in Figure 56.

```
# Format the HDFS
$hadoop namenode -format
# Start the namenode and the data nodes as cluster
$ start-dfs.sh
# List of files in a directory using ls command
$ $HADOOP_HOME/bin/hadoop fs -ls <args>

# Inserting Data into HDFS
# 1. Create an input directory
$ $HADOOP_HOME/bin/hadoop fs -mkdir /user/input
# 2. Transfer and store a data file from local systems to the
# Hadoop file system using the put command
$ $HADOOP_HOME/bin/hadoop fs -put /home/file.txt /user/input
# 3. Verify the file using ls command
$ $HADOOP_HOME/bin/hadoop fs -ls /user/input

# Retrieving data from HDFS
# 1. View the data from HDFS using cat command
$ $HADOOP_HOME/bin/hadoop fs -cat /user/output/outfile
# 2. Get the file from HDFS to the local file system using get command
$ $HADOOP_HOME/bin/hadoop fs -get /user/output/ /home/hadoop_tp/

# Shutting Down the HDFS
$ stop-dfs.sh
```

Figure 56: Useful commands for HDFS

There are many good tutorials on learning HDFS. One can be found on YouTube [81].

### 7.2.7 Apache HBase

Apache HBase is an open source project modeled after Google's Big Table [82]. It is written in Java, and is a part of Apache Hadoop. It runs on top of HDFS. It is a non-relational distributed database, unlike RDBMSs like MySQL. It is written in Java.

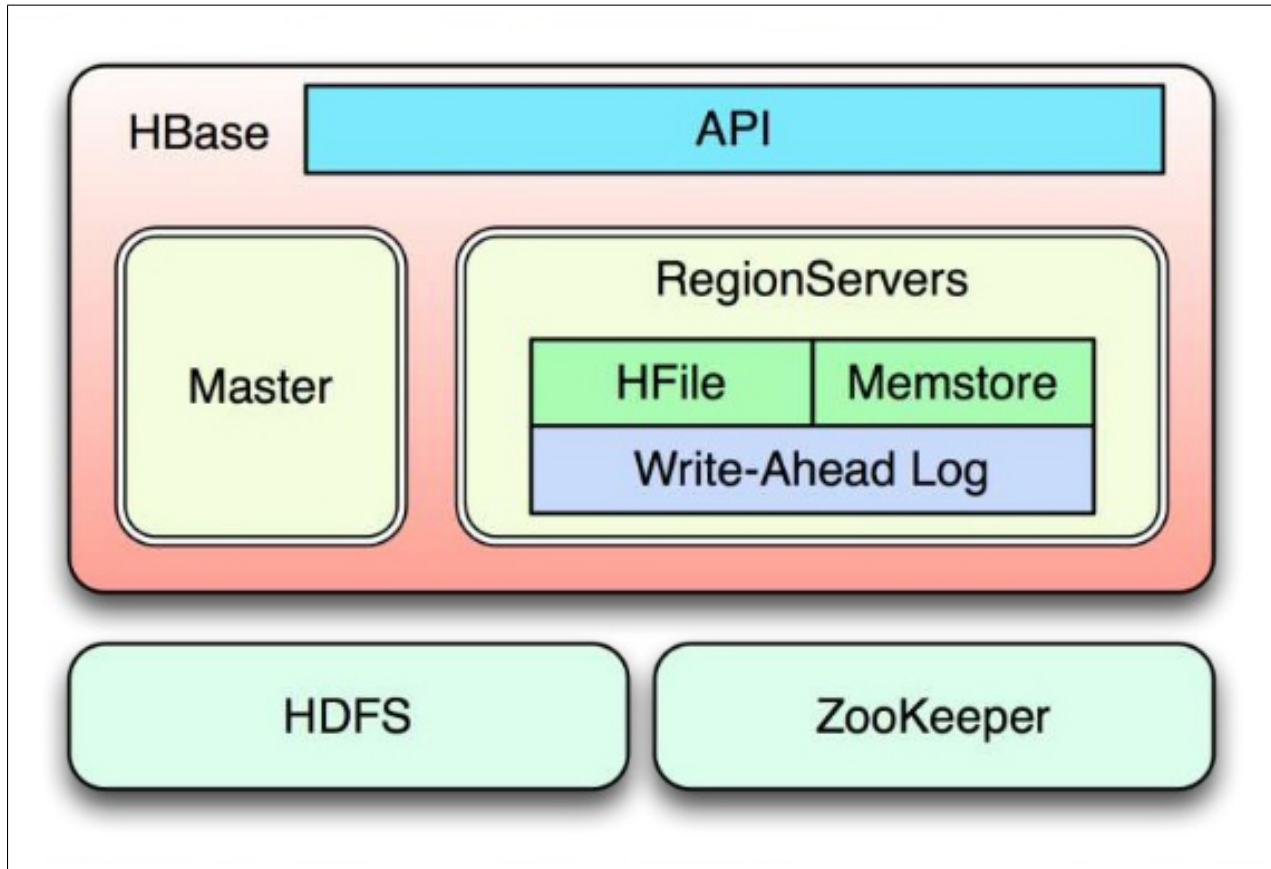


Figure 57: HBase architecture [21]

Some characteristics and advantages of using HBase are as follows.

1. Fault tolerant - HBase duplicates data across the cluster. Thus, if one node goes down, data is not lost. It automatically takes care of load-balancing of tables. It also provides high availability meaning that it has a very high up-time ratio.
2. Fast Operations - The lookup time in HBase tables is very fast and almost real time. It also does in-memory caching, which speeds up data transfers as well.
3. Flexible - it supports a wide array of use cases. It is also easy to read the metrics of HBase using something like Ganglia [83].
4. Usability - it also provides Java APIs as well as REST gateway APIs.

It makes most sense to use HBase when one has billions of rows and columns in a table. In the case of small data sets, it makes more sense to use a RDBMS that runs on a single server. HBase stores key-value pairs in a columnar fashion and provides low latency access to small amounts of data within a big data set.

There are certain limitations on using HBase, as described below:

1. It can not be used for transactional applications or where there is a need for relational analysis.
2. You cannot communicate with it in the manner you would normally communicate with a SQL database. It does not support cross joins, etc.
3. You typically cannot use HBase with more complicated query patterns where you are using lots of ‘joins’ to extract data.

### 7.2.8 Apache HBase installation & operation

HBase is a column-oriented NoSQL database that runs on top of HDFS. It is well-suited for sparse datasets, which are common in dealing with big data, and provides for the storing of databases with billions of rows and millions of columns. Much in the same way HDFS and Hadoop have master nodes that keep track of slaves, HBase utilizes a master node to store portions of tables across a network of machines [84].

Columns in HBase are grouped together into column families that are stored together on the filesystem. System tuning and storage specifications are done at the column family level, and families must be declared up front during the schema definition. Column families should logically group columns that likely have a similar access pattern [85].

1. **Installation:** Follow the steps on the website [86] to download and install HBase.
2. **Operation:** There are many useful things that you can do using HBase. We have listed some of the basic ones below in Figure 58.

```

command for HBase
# Starting HBase Shell
cd /usr/localhost/
cd Hbase
# Provides the status of HBase, for example, the number of servers
status
# Provides the version of HBase being used
version
# Creating a Table using HBase Shell
create '<table name>', '<column family>'
# Listing a Table using HBase Shell
hbase(main):001:0 > list

# Note: To delete a table or change its settings, you need
# to first disable the table using the disable command. You
# can re-enable it using the enable command.

# Disabling a Table using HBase Shell
disable '<table name>'
# Enabling a Table using HBase Shell
enable '<table name>'
# Reading Data using HBase Shell
get '<table name>', 'row1'
# Reading a Specific Column
get '<table name>', 'rowid', {COLUMN => 'column family:column name '}
# Inserting Data using HBase Shell Using the put command
put '<table name>', 'row1', '<colfamily:colname>', '<value>'
# Deleting a Specific Cell in a Table Using the delete command
delete '<table name>', '<row>', '<column name >', '<time stamp>'
# Scanning using HBase Shell using the scanning command
scan '<table name>'

```

Figure 58: Useful commands for HBase

There are many good tutorials on learning HBase. One can be found on YouTube [87].



### 7.2.9 Apache Sqoop

Sqoop is a tool designed to transfer data between Hadoop and relational database servers. It is used to import data from relational databases such as MySQL or Oracle to Hadoop HDFS, and export from the Hadoop file system to relational databases. It is provided by the Apache Software Foundation [88].

The Sqoop import tool imports individual tables from an RDBMS to HDFS. Each row in a table is treated as a record in HDFS. All records are stored as text data in text files or as binary data in Sequence files. The export tool exports a set of files from HDFS back to an RDBMS. The files given as input to Sqoop contain records, which are called as rows in the table. Those are read and parsed into a set of records and delimited with a user-specified delimiter [88].

One of the main benefits of Sqoop is that it automatically implements parallel data transfers with built-in fault-tolerance, which is important when transferring even moderate amounts of data.

There are many good tutorials on learning Sqoop. One which we found useful is a Sqoop user guide [89].

### 7.2.10 Apache Sqoop installation & operation

Apache Sqoop is a tool designed for efficiently transferring bulk data between Apache Hadoop and structured datastores such as relational databases. In this project, we will use Sqoop to transfer data from relational DB to HDFS.

1. **Installation:** Follow the steps on the website [90] to download and install Sqoop.
2. **Operation:** Figure 59 shows three examples of how to import and export data to HDFS.

```

## Three examples

# Import emp_add table data into '/queryresult' directory
$ sqoop import \
--connect jdbc:mysql://localhost/userdb \
--username root \
--table emp_add \
--m 1 \
--target-dir /queryresult

# Import all the tables from the userdb database
$ sqoop import-all-tables \
--connect jdbc:mysql://localhost/userdb \
--username root

# Export the table data to the employee table in db database
$ sqoop export \
--connect jdbc:mysql://localhost/db \
--username root \
--table employee \
--export-dir /emp/emp_data

```

Figure 59: Examples of importing and exporting data to HDFS using Sqoop

There are many good tutorials on learning Sqoop. One can be found on YouTube [91].

### 7.2.11 Apache Pig

Pig is an Apache project that is used for creating programs that use Hadoop or Spark [62] to run. Users can make use of the platform through the Pig Latin language designed for it. Pig Latin scripts can run in MapReduce fashion, Apache Tez, or Apache Spark [23]. Pig Latin can also be extended using User Defined Functions. These functions can be written in JavaScript, Python, Ruby, or Groovy, and then called directly from the language.

Historically, Pig was developed at Yahoo Research Labs in 2006 to run MapReduce jobs on very large datasets. In 2007, the Pig code repository moved into the Apache Software Foundation.

There are many advantages of using Pig. Some of them are given below [92].

- Ease of programming - coding in Pig is relatively simple, and it is easy to achieve parallel execution of simple tasks. Pig code is easy to maintain, write, and understand.
- Optimization opportunities - the tasks are optimized automatically, which allows users to focus on semantics rather than efficiency.
- Extensibility - just like many other high-level programming languages, Pig also allows custom user functions to do specific kinds of data processing.

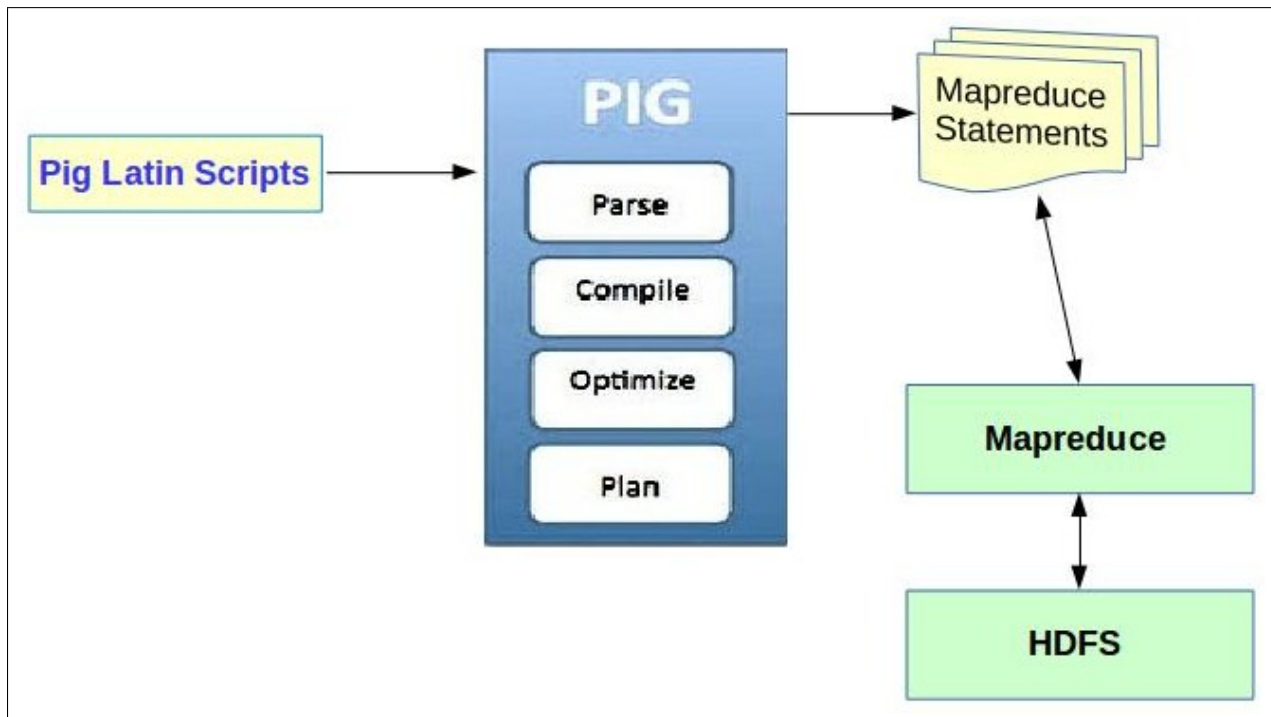


Figure 60: Workflow of Pig [22]

### 7.2.12 Apache Pig installation & operation

Pig is used in this project for data cleaning and data transfer.

1. **Installation:** Follow the steps mentioned on the Tutorials Point website [93] to download and install Pig.
2. **Operation:** There are many useful things that you can do using Pig. We have listed some of the basic ones below. We also include a simple word count example using Pig in Figure 62.

```
command for Pig

## LOAD: load data into Apache Pig from the file system.
Relation_name = LOAD 'Input file path' USING function as schema;
## DUMP: Dumps or displays results to screen.
A = LOAD 'student' AS (name:chararray, age:int, gpa:float);
DUMP A;
## DESCRIBE: Returns the schema of an alias.
A = LOAD 'student' AS (name:chararray, age:int, gpa:float);
DESCRIBE A;
## FILTER: Selects tuples from a relation based on some condition.
A = LOAD 'data' AS (f1:int,f2:int,f3:int);
X = FILTER A BY f3 == 3;
## SPLIT: Partitions a relation into two or more relations.
A = LOAD 'data' AS (f1:int,f2:int,f3:int);
SPLIT A INTO X IF f1<7, Y IF f2==5, Z IF (f3<6 OR f3>6);
## STORE: Stores or saves results to the file system.
A = LOAD 'data' AS (a1:int,a2:int,a3:int);
STORE A INTO 'myoutput' USING PigStorage('*');
## GROUP: Groups the data in one or multiple relations.
A = load 'student' AS (name:chararray,age:int,gpa:float);
B = GROUP A BY age;
## FOREACH: Generates data transformations based on columns of data.
X = FOREACH A GENERATE f1;
```

Figure 61: Useful commands for Pig

To learn more about Pig, please watch this video tutorial by HortonWorks [94].

```

input_lines = LOAD '/tmp/my-copy-of-all-pages-on-internet' AS (line:chararray);

-- Extract words from each line and put them into a pig bag
-- datatype, then flatten the bag to get one word on each row
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;

-- filter out any words that are just white spaces
filtered_words = FILTER words BY word MATCHES '\\w+';

-- create a group for each word
word_groups = GROUP filtered_words BY word;

-- count the entries in each group
word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS count, group AS word;

-- order the records by count
ordered_word_count = ORDER word_count BY count DESC;
STORE ordered_word_count INTO '/tem/number-of-words-of-internet';

```

Figure 62: Simple word count example using Pig [23]

### 7.2.13 Pt-archiver installation & operation

Pt-archiver [45] is used in this project for archiving the rows from the CollectDB to the ArchiveDB.

1. **Installation:** pt-archiver comes in a toolkit provided by Percona. Its installation is depicted in Figure 63.

Visit <http://www.percona.com/software/percona-toolkit/> to download the latest release of Percona Toolkit. Or, get the latest release from the command line:

```
wget percona.com/get/percona-toolkit.tar.gz
```

```
wget percona.com/get/percona-toolkit.rpm
```

```
wget percona.com/get/percona-toolkit.deb
```

You can also get individual tools from the latest release:

```
wget percona.com/get/TOOL
```

Replace TOOL with the name of any tool.

Figure 63: Installing pt-archiver

2. **Operation:** There are many options that you can use with pt-archiver. Two simple examples are shown in Figure 64.

In the first example, pt-archiver archives all rows from the “oltp server” using database “test”, and table “tbl1” to the destination server “olap server”. In addition to archiving the rows to a new table, it also saves the archived rows in a file. Note that the rows from the source database will be deleted in this case after transferring them to the destination database.

In the second example, pt-archiver deletes all the orphan rows left in the source database using a “WHERE” clause. The users can customize this clause as they see fit. See Figure 64 for two working examples.

```
pt-archiver [OPTIONS] --source DSN --where WHERE
```

**pt-archiver** nibbles records from a MySQL table. The `--source` and `--dest` arguments use DSN syntax; if `COPY` is yes, `--dest` defaults to the key’s value from `--source`.

### Examples

Archive all rows from `oltp_server` to `olap_server` and to a file:

```
pt-archiver --source h=oltp_server,D=test,t=tbl --dest h=olap_server \
--file '/var/log/archive/%Y-%m-%d-%D.%t' \
--where "1=1" --limit 1000 --commit-each
```

Purge (delete) orphan rows from child table:

```
pt-archiver --source h=host,D=db,t=child --purge \
--where 'NOT EXISTS(SELECT * FROM parent WHERE col=child.col)'
```

Figure 64: Simple usage of pt-archiver

## 7.2.14 csv2avro

To convert CSV files to Avro files, we are making using of an open source tool called csv2avro[31]. We decided on this tool after many failed attempts to convert CSV files to Avro files using PiggyBank[61], and avroStorage[95].

## 7.2.15 csv2avro Installation & Operation

1. **Installation:** You need to clone the git repository from [31], and install it on your system. You do not need to compile the package from source unless you know what you are doing. See Figure 65 for details.

```
[faiz89@viz1: ~]$ git clone https://github.com/sspinc/csv2avro.git
Cloning into 'csv2avro'...
remote: Counting objects: 1246, done.
remote: Total 1246 (delta 0), reused 0 (delta 0), pack-reused 1246
Receiving objects: 100% (1246/1246), 150.30 KiB | 0 bytes/s, done.
Resolving deltas: 100% (660/660), done.
Checking connectivity... done.
[faiz89@viz1: ~]$ sudo gem install csv2avro
[sudo] password for faiz89:
Successfully installed csv2avro-1.3.1
Parsing documentation for csv2avro-1.3.1
Done installing documentation for csv2avro after 0 seconds
1 gem installed
```

Figure 65: Installing csv2avro

2. **Operation:** There are some extra flags that you can pass while running csv2avro. Figure 66 shows the flags that we passed while running this tool.

```
#!/bin/bash
csv2avro \
  --schema schema.avsc \
  --delimiter "\t" \
  --line-ending "\n" \
  --bad-rows ~/avro_data/rows3.bad \
  ~/avro_data/2016-11-22-test_faiz.z_312.csv
```

Figure 66: Using csv2avro

The schema file needs to be passed, along with the delimiter and the line-ending character. The flag “--bad-rows” keeps a record of any row that can not be parsed because of the presence of any invalid characters.

## 7.3 Project installation

We installed KVM [24] on one of our lab machines to create a virtual machine. We followed the steps in Figures 67, 68, and 69.

### Install Necessary Packages

For the following setup, we will assume that you are deploying KVM on a server, and therefore do not have any X server on the machine.

You need to install a few packages first:

#### Lucid (10.04) or later

```
$ sudo apt-get install qemu-kvm libvirt-bin ubuntu-vm-builder bridge-utils
```

#### Karmic (9.10) or earlier

```
$ sudo aptitude install kvm libvirt-bin ubuntu-vm-builder bridge-utils
```

1. libvirt-bin provides libvirtd which you need to administer qemu and kvm instances using libvirt
2. qemu-kvm (kvm in Karmic and earlier) is the backend
3. ubuntu-vm-builder powerful command line tool for building virtual machines
4. bridge-utils provides a bridge from your network to the virtual machines

You might also want to install virt-viewer, for viewing instances.

### Add Users to Groups

#### Karmic (9.10) and later (but not 14.04 LTS)

You need to ensure that your username is added to the group libvirtd:

```
$ sudo adduser `id -un` libvirtd
Adding user '<username>' to group 'libvirtd' ...
```

After this, **you need to relogin** so that your user becomes an effective member of the libvirtd group. The members of this group can run virtual machines. (You can also 'newgrp kvm' in a terminal, but this will affect only that terminal.)

#### Releases prior to Karmic (9.10)

You need to ensure that your username is added to the groups: kvm and libvirtd.

To check:

```
$ groups
adm dialout cdrom floppy audio dip video plugdev fuse lpadmin admin sambashare kvm libvirtd
```

Figure 67: Instructions for installing KVM: Part 1 [24]



To add your <username> to the groups:

```
$ sudo adduser `id -un` kvm
Adding user '<username>' to group 'kvm' ...
$ sudo adduser `id -un` libvirt
Adding user '<username>' to group 'libvirt' ...
```

After the installation, **you need to relogin** so that your user becomes an effective member of kvm and libvirt user groups. The members of this group can run virtual machines.

## Verify Installation

You can test if your install has been successful with the following command:

```
$ virsh list --all
Id Name                               State
-----
$
```

If on the other hand you get something like this:

```
$ virsh list --all
libvir: Remote error : Permission denied
error: failed to connect to the hypervisor
$
```

Something is wrong (e.g. you did not relogin) and you probably want to fix this before you move on. The critical point here is whether or not you have write access to /var/run/libvirt/libvirt-sock.

The sock file should have permissions similar to:

```
$ sudo ls -la /var/run/libvirt/libvirt-sock
srwxrwx--- 1 root libvirt 0 2010-08-24 14:54 /var/run/libvirt/libvirt-sock
```

Also, /dev/kvm needs to be in the right group. If you see:

```
$ ls -l /dev/kvm
crw-rw----+ 1 root root 10, 232 Jul  8 22:04 /dev/kvm
```

You might experience problems when creating a virtual machine. Change the device's group to kvm/libvirt instead:

```
sudo chown root:libvirt /dev/kvm
```

Figure 68: Instructions for installing KVM: Part 2 [24]

Now you need to either **relogin** or restart the kernel modules:

```
rmmod kvm
modprobe -a kvm
```

## Optional: Install virt-manager (graphical user interface)

If you are working on a desktop computer you might want to install a GUI tool to manage virtual machines.

```
$ sudo apt-get install virt-manager
```

Figure 69: Instructions for installing KVM: Part 3 [24]

Once we had KVM running, we downloaded the Cloudera image, which was to be used by the virtual machine.

```

[[faiz89@viz1 Some_Install_Scripts]$ wget http://hadoop.dlib.vt.edu/cloudera_vm/cloudera-quickstart-vm-5.3.0-0-kvm.7z
--2016-09-20 22:50:33-- http://hadoop.dlib.vt.edu/cloudera_vm/cloudera-quickstart-vm-5.3.0-0-kvm.7z
Resolving hadoop.dlib.vt.edu (hadoop.dlib.vt.edu)... 128.173.49.66
Connecting to hadoop.dlib.vt.edu (hadoop.dlib.vt.edu)|128.173.49.66|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3326631605 (3.1G) [application/x-troff-man]
Saving to: 'cloudera-quickstart-vm-5.3.0-0-kvm.7z'

[cloudera-quickstart-vm-5.3.0-0-kvm.7z

```

Figure 70: Downloading the Cloudera image for KVM

Next, we spun up a VM using virsh.

```

[[faiz89@viz1 Some_Install_Scripts]$ cat create_vm2.py
#!/usr/bin/python

import subprocess

retcode = subprocess.call("virt-install "
    "--connect qemu:///system "
    "--n cloudera-vm "
    "--r 8192 "
    "--vcpus=4 "
    "--disk path=/home/faiz89/Images/cloudera-quickstart-vm-5.3.0-0-kvm/\
cloudera-quickstart-vm-5.3.0-0-kvm.qcow2,device=disk,bus=virtio,format=qcow2"
    "--ram 8192 "
    "--vnc "
    "--noautoconsole "
    "--import", shell=True)

if retcode == 0:
    print("Successfully created a VM :)")
else:
    print("VM creation failed.")

```

Figure 71: Creating a VM using KVM

```

[cloudera@quickstart ~]$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 171
Server version: 5.1.66 Source distribution

Copyright (c) 2000, 2012, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| cm |
| firehose |
| hue |
| metastore |
| mysql |
| oozie |
| retail_db |
| sentry |
| test_small_data |
+-----+
10 rows in set (0.01 sec)

mysql> use test_small_data;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql>

```

Figure 72: Logging into the MySQL database on the Cloudera VM

We then loaded the dump of SQL data we received from Sunshin into our MySQL database running on our virtual machine. We created a test database called “test\_small\_data”.

```

1  mysqldump --all-databases > dump-$( date '+%Y-%m-%d_%H-%M-%S' ).sql -u root -p

```

Figure 73: Dumping the test SQL database into MySQL

## 7.4 Software versions

We provide a list of the software versions we used for development.

- CentOS, release 6.7
- OpenJDK, version 1.7.0\_101
- PiggyBank, version 0.12.0
- Apache Pig, version 0.12.0-cdh5.6.0
- Stanford CoreNLP, version 3.4.1 (required to work with Java 7)
- HBase Java API, version 1.2.3
- avro-tools, version 1.8.1
- MySQL, version 5.7.16
- Hadoop version 2.6.0-cdh5.6.0
- pt-archiver, version 2.2.19
- csv2avro, version 1.3.1
- Python 2.6.6

### 7.4.1 MySQL to HDFS incremental update

All the code that deals with the incremental update feature from the MySQL server to the HDFS server can be found in the sub-folder called “Tweets\_Update\_Incremental\_HDFS” in our GitHub repository that can be found in [64]. The scripts are fairly well documented with comments in them, but some things to take care of while running the scripts are:

1. We are using pt-archiver to generate the text file for the tweets. If pt-archiver undergoes some update, make sure to update the version of the pt-archiver installed on your system as well. If some old flags are replaced by newer ones then the script called “transfer\_data\_using\_pt-archiver.bash” will need to be altered.
2. Our script called “cleanup\_tweets.bash” removes all the non-ASCII, newline, double quote, and comma characters. If the tweets have some other unknown character, this script will fail to process those rows. In that case, a file called bad.rows is generated with the line number that has the invalid character. In such a scenario, our script will need to be augmented to take care of the newly found invalid characters.
3. Each Avro file needs to be put on a specific location on HDFS. In our script called, “convert\_csv\_avro.bash”, we have hard-coded the base path on the HDFS system as “/collections/tweets-705”. In case the base path changes on the HDFS system, this hard-coded value in the script will also need to be changed.

4. We use the open source tool called `csv2avro` to convert CSV files into Avro files. If `csv2avro` gets updated, make sure to update it on your system as well. Our script called “`convert_csv_avro.bash`” may also need to be altered in case of any major update to `csv2avro`.
5. Avro files need to be merged on the HDFS system. We use `avro-tools` version 1.8.1. We have not tested our script with any other versions. We believe newer versions should not break it, but if they do, our script called, “`merge_avro.bash`” will also need to be altered.

#### 7.4.2 HDFS to HBase incremental update

There are a number of improvements that can be made to the scripts in our tweet processing pipeline. These include:

1. Making the scripts more flexible: we currently have hard-coded values (for example, the directory that will eventually be concatenated with the collection number that we pass some scripts). It would be beneficial to make these arguments that can be passed to the scripts
2. Implementing error handling via checking error codes.
3. Re-implementing the pipeline using different technologies, such as Apache Spark, for faster processing [62].

It is likely that individuals wishing to use our code will need to adapt at least some part of it to their own system. For example, the path to the Avro file that the individuals use might be different, or they might not be using collection numbers in their own work in the manner that we are (Figure 74).

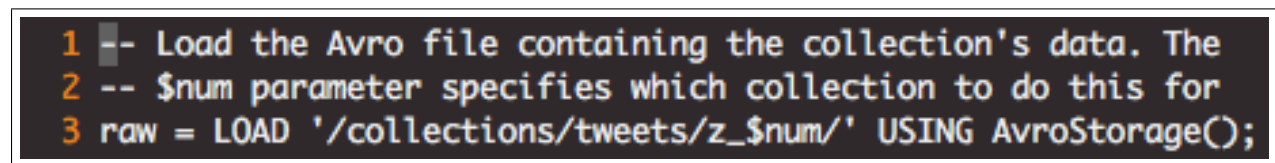


Figure 74: Example of hard-coded filename combined collection number parameters

A final thing to be aware of is that our processing pipeline assumes that the Avro files are following the schema given by 19.

#### 7.4.3 Social network

There are some improvements that can be made to the scripts for building social network and we list some of those below:

1. For computing the importance factor for URLs, we just use the number of times the URL is mentioned in a tweet. However, we believe that it would be more useful if we were able to include some content-based features in the calculation for the importance factor. Alternatively, employing an algorithm like PageRank might allow us to garner a greater global insight of the entity’s importance.

2. We could improve the method for assigning the weights for each feature used in calculating the importance factor of users and tweets. Right now we just assign the weights manually based on our understanding, which is shown in Figure 75, but it would be better to do this in a more data-driven fashion.

```
76 def calculate_user_importance(user_info):
77     im = []
78
79     for user in user_info:
80         x = user[4] * 0.25 + user[5] * 0.25 + user[6] * 0.10 + user[7] * 0.25 + user[8] * 0.15
81         im.append((x))
82
83     min_max_scaler = preprocessing.MinMaxScaler()
84     imp = min_max_scaler.fit_transform(im)
85     i = 0
86     for user in user_info:
87         user.append((imp[i]))
88         i += 1
89
90     return user_info
```

Figure 75: The assignment of weights for calculating the user importance factors

3. At the moment, our code is dependent on Tweepy, so if Tweepy changes and we want to continue using the updated version or take advantage of the new features, we might have to account for that. Removing this dependence might be useful in a long-term project such as the IDEAL/GETAR projects.

In case you have any questions that remain unanswered, feel free to contact us. We would be happy to help. Our contact details can be found on the first page of this report.

## 7.5 File inventory

The following is a list of files that we have included in our VTechWorks submission.

- Final project report
- Final project presentation
- Final project code: a full copy of the code we have developed for this project is available at:

<https://github.com/mitchwagner/CMT>

## 8 Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. IIS-1619028 and Grant No. IIS-1319578.

## 9 Appendix

### 9.1 Table of acronyms

Table 5: Table of acronyms used in this report

| <b>Acronym</b> | <b>Meaning</b>                                 |
|----------------|--|
| IDEAL          | Integrated Digital Event Archiving and Library |
| GETAR          | Global Event and Trend Archive Research        |
| CLA            | Classification Team                            |
| CMT            | Collection Management Team (Tweets)            |
| CMW            | Collection Management Team (Web Pages)         |
| CTA            | Clustering and Topic Analysis Team             |
| FE             | Front End Team                                 |
| SOLR           | Solr Team                                      |
| DBMS           | Database Management System                     |
| RDBMS          | Relational Database Management System          |
| YTK            | YourTwapperKeeper                              |



## References

- [1] E. A. Fox, K. Hanna, A. L. Kavanaugh, S. D. Sheetz, D. J. Shoemaker, *et al.*, “Integrated Digital Event Archiving and Library (IDEAL).” <http://grantome.com/grant/NSF/IIS-1319578>, 2014. (accessed 12-19-2016).
- [2] “Global Event and Trend Archive Research (GETAR).” <http://www.eventsarchive.org/sites/default/files/GETARsummaryWeb.pdf>, 2016.
- [3] “HDFS Architecture Guide.” [https://hadoop.apache.org/docs/r1.2.1/hdfs\\_design.html](https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html), 2016. (accessed 12-19-2016).
- [4] Apache, “Welcome to Apache HBase.” <http://hbase.apache.org>, 2016. (accessed 12-19-2016).
- [5] S. Lee, “Data Flow Diagram of the IDEAL Infrastructure.” CS 5604 Class Website, 2016.
- [6] Y. Ma and D. Nan, “Collection Management for IDEAL.” <http://vtechworks.lib.vt.edu/handle/10919/70930>, 2016. (accessed 12-19-2016).
- [7] Oracle, “Database Backup.” [www.oracle.com](http://www.oracle.com), 2016. (accessed 12-19-2016).
- [8] “Full backup of data.” <http://www.databasethink.com/mysql/backup/help/backup-type/images/full-backup.jpg>, 2016. (accessed 12-19-2016).
- [9] “Commvault Advanced Backup.” [https://documentation.commvault.com/commvault/v10/article?p=products/db2/backup\\_adv.htm](https://documentation.commvault.com/commvault/v10/article?p=products/db2/backup_adv.htm), 2016. (accessed 12-19-2016).
- [10] “Differential backup.” <http://sysinfotools.com/blog/wp-content/uploads/2013/07/Differential-backup.png>, 2016. (accessed 12-19-2016).
- [11] “Different types of data backups.” <http://searchdatabackup.techtarget.com/definition/differential-backup>, 2016. (accessed 12-19-2016).
- [12] W. Wong, W. Liu, and M. Bennamoun, “Enhanced integrated scoring for cleaning dirty texts,” *arXiv preprint arXiv:0810.0332*, 2008.
- [13] “Touch Graph.” <http://www.touchgraph.com/news>, 2016. (accessed 12-19-2016).
- [14] B. Furht, *Handbook of social network technologies and applications*. Springer Science & Business Media, 2010.
- [15] T. S. Vishwasrao, Saket and L. Tang, “CS5604: Clustering and Social Networks for IDEAL.” <http://vtechworks.lib.vt.edu/handle/10919/70947>, 2016. (accessed 12-19-2016).
- [16] Wikipedia, “PageRank Algorithm.” <https://en.wikipedia.org/wiki/PageRank>, 2016. (accessed 12-19-2016).
- [17] “Free Source Code – Twitter Database Server: MySQL Database Schema.” [http://140dev.com/tutorial\\_images/twitter\\_database.png](http://140dev.com/tutorial_images/twitter_database.png), 2016. (accessed 12-19-2016).

- [18] “Sun and MySQL: How It Stacks Up for Developers.” <http://www.oracle.com/technetwork/articles/java/mysql-acq-139875.html>, 2016. (accessed 12-19-2016).
- [19] “Hadoop Architecture.” [https://opensource.com/sites/default/files/resize/images/life-uploads/hadoop-HighLevel\\_hadoop\\_architecture-640x460.png](https://opensource.com/sites/default/files/resize/images/life-uploads/hadoop-HighLevel_hadoop_architecture-640x460.png), 2016. (accessed 12-19-2016).
- [20] “HDFS Architecture Guide.” <http://hadoop.apache.org/docs/r1.2.1/images/hdfsarchitecture.gif>, 2013. (accessed 12-19-2016).
- [21] “HBase Architecture.” <http://www.cloudera.com>, 2016. (accessed 12-19-2016).
- [22] “Pig High-level View.” [http://hadoopmag.com/wp-content/uploads/2014/04/PIG-2\\_html\\_m212e6cdd.png](http://hadoopmag.com/wp-content/uploads/2014/04/PIG-2_html_m212e6cdd.png), 2016. (accessed 12-19-2016).
- [23] “Pig (programming tool).” [https://en.wikipedia.org/wiki/Pig\\_\(programming\\_tool\)](https://en.wikipedia.org/wiki/Pig_(programming_tool)), 2016. (accessed 12-19-2016).
- [24] “Kernel Virtual Machine.” [http://www.linux-kvm.org/page/Main\\_Page](http://www.linux-kvm.org/page/Main_Page), 2016. (accessed 12-19-2016).
- [25] “yourTwapperKeeper.” <https://github.com/540co/yourTwapperKeeper>, 2013. (accessed 12-19-2016).
- [26] “Social Feed Manager: Helping researchers and archivists build social media collections.” <http://gwu-libraries.github.io/sfm-ui/>, 2016. (accessed 12-19-2016).
- [27] “Digital Methods Initiative - Twitter Capture and Analysis Toolset.” <https://github.com/digitalmethodsinitiative/dmi-tcat>, 2016. (accessed 12-19-2016).
- [28] “Blacklight - A multi-institutional open-source collaboration building a better discovery platform framework.” <http://projectblacklight.org/>, 2016. (accessed 12-19-2016).
- [29] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, “The Stanford CoreNLP natural language processing toolkit,” in *Association for Computational Linguistics (ACL) System Demonstrations*, pp. 55–60, 2014.
- [30] “VTechWorks.” <https://vtechworks.lib.vt.edu/>, 2016. (accessed 12-19-2016).
- [31] “csv2avro - Convert CSV files to Avro.” <https://github.com/sspinc/csv2avro>, 2016. (accessed 12-19-2016).
- [32] “Comparison of MySQL database engines.” [https://en.wikipedia.org/wiki/Comparison\\_of\\_MySQL\\_database\\_engines](https://en.wikipedia.org/wiki/Comparison_of_MySQL_database_engines), 2016. (accessed 12-19-2016).
- [33] MySQL, “Converting Tables from MyISAM to InnoDB.” <http://dev.mysql.com/doc/refman/5.7/en/converting-tables-to-innodb.html>, 2016. (accessed 12-19-2016).
- [34] A. Khurana, “Introduction to HBase Schema Design.” [https://www.usenix.org/system/files/login/articles/login1210\\_khurana.pdf](https://www.usenix.org/system/files/login/articles/login1210_khurana.pdf), October 2012. (accessed 12-19-2016).

- [35] Techopedia, “Database Backup.” <https://www.techopedia.com/definition/29388/database-backup>, 2016. (accessed 12-19-2016).
- [36] “The MyISAM Storage Engine.” <http://dev.mysql.com/doc/refman/5.7/en/myisam-storage-engine.html>, 2016. (accessed 12-19-2016).
- [37] “The InnoDB Storage Engine.” <http://dev.mysql.com/doc/refman/5.7/en/innodb-storage-engine.html>, 2016. (accessed 12-19-2016).
- [38] Wikipedia, “Hurricane Katrina.” [https://en.wikipedia.org/wiki/Hurricane\\_Katrina](https://en.wikipedia.org/wiki/Hurricane_Katrina), 2016. (accessed 12-19-2016).
- [39] Q. Li and H. Xu, “Research on the backup mechanism of Oracle database,” in *Environmental Science and Information Application Technology, 2009. ESIAT 2009. International Conference on*, vol. 2, pp. 423–426, IEEE, 2009.
- [40] MySQL, “MySQL 5.7 Manual.” <http://dev.mysql.com/doc/refman/5.7/en>, 2016. (accessed 12-19-2016).
- [41] Wikipedia, “Differential backup.” [https://en.wikipedia.org/wiki/Differential\\_backup](https://en.wikipedia.org/wiki/Differential_backup), 2016. (accessed 12-19-2016).
- [42] Percona, “Percona - the database performance experts.” <https://www.percona.com/>, 2016. (accessed 12-19-2016).
- [43] “Maatkit Toolkit.” <https://sourceforge.net/projects/maatkit/>, 2016. (accessed 12-19-2016).
- [44] “Aspersa Toolkit.” <https://github.com/true/aspersa-mirror>, 2016. (accessed 12-19-2016).
- [45] “pt-archiver - Percona Toolkit.” <https://www.percona.com/doc/percona-toolkit/2.1/pt-archiver.html>, 2016. (accessed 12-19-2016).
- [46] “Introduction to Natural language Processing.” <http://bdewilde.github.io/blog/blogger/2013/04/16/intro-to-natural-language-processing-2/>, 2013. (accessed 12-19-2016).
- [47] F. J. Damerau, “A technique for computer detection and correction of spelling errors,” *Communications of the ACM*, vol. 7, no. 3, pp. 171–176, 1964.
- [48] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions and reversals,” in *Soviet physics doklady*, vol. 10, p. 707, 1966.
- [49] K. Kukich, “Techniques for automatically correcting words in text,” *ACM Computing Surveys (CSUR)*, vol. 24, no. 4, pp. 377–439, 1992.
- [50] M. S. Hearst, “A simple algorithm for identifying abbreviation definitions in biomedical text,” 2003.

- [51] Y. Park and R. J. Byrd, “Hybrid text mining for finding abbreviations and their definitions,” in *Proceedings of the 2001 conference on empirical methods in natural language processing*, pp. 126–133, 2001.
- [52] A. Mikheev, “Periods, capitalized words, etc.,” *Computational Linguistics*, vol. 28, no. 3, pp. 289–318, 2002.
- [53] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.
- [54] “How we analyzed Twitter social media networks with NodeXL.” <http://www.pewinternet.org/files/2014/02/How-we-analyzed-Twitter-social-media-networks.pdf>, 2016. (accessed 12-19-2016).
- [55] A. Mtibaa, A. Chaintreau, J. LeBrun, E. Oliver, A.-K. Pietilainen, and C. Diot, “Are you moved by your social network application?,” in *Proceedings of the first workshop on Online social networks*, pp. 67–72, ACM, 2008.
- [56] L. Page, S. Brin, R. Motwani, and T. Winograd, “The pagerank citation ranking: Bringing order to the web.” Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.
- [57] Google, “Facts about Google and competition.” <https://web.archive.org/web/20111104131332/http://www.google.com/competition/howgooglesearchworks.html>, 2016. (accessed 12-19-2016).
- [58] S. Brin and L. Page, “Reprint of: The anatomy of a large-scale hypertextual web search engine,” *Computer networks*, vol. 56, no. 18, pp. 3825–3833, 2012.
- [59] “Text Wrangler.” <http://www.barebones.com/products/TextWrangler/>, 2016. (accessed 12-19-2016).
- [60] Apache, “Apache Avro.” <https://avro.apache.org/>, 2016. (accessed 12-19-2016).
- [61] “Piggy Bank - User Defined Pig Functions.” <https://cwiki.apache.org/confluence/display/PIG/PiggyBank>, 2016.
- [62] Apache, “Apache Spark.” <http://spark.apache.org/>, 2016. (accessed 12-19-2016).
- [63] A. A. Hagberg, D. A. Schult, and P. J. Swart, “Exploring network structure, dynamics, and function using NetworkX,” in *Proceedings of the 7th Python in Science Conference (SciPy2008)*, (Pasadena, CA USA), pp. 11–15, Aug. 2008.
- [64] F. Abidi, S. Fan, and M. Wagner, “CMT Team’s Codebase on GitHub.” <https://github.com/mitchwagner/CMT>, 2016. (accessed 12-19-2016).
- [65] “Apache Avro 1.8.1 Getting Started.” <https://avro.apache.org/docs/1.8.1/gettingstartedjava.html>, 2016. (accessed 12-19-2016).
- [66] Gradle, “Getting started.” <https://gradle.org/getting-started-gradle-java/>, 2016. (accessed 12-19-2016).

- [67] “Tweepy: Twitter for Python!” <https://github.com/tweepy/tweepy>, 2016. (accessed 12-19-2016).
- [68] “RDBMS and Graphs, Relational vs. Graph Data Modeling.” <https://neo4j.com/blog/rdbms-vs-graph-data-modeling/>, 2014. (accessed 12-19-2016).
- [69] “Top reason to use MySQL.” <https://www.mysql.com/why-mysql/topreasons.html>, 2016. (accessed 12-19-2016).
- [70] The bad tutorials. [https://www.youtube.com/watch?v=-th0n1NKJew&list=PL\\_RGaFnxSHWr\\_6xTfF2FrIw-NA0o3iWMy](https://www.youtube.com/watch?v=-th0n1NKJew&list=PL_RGaFnxSHWr_6xTfF2FrIw-NA0o3iWMy), 2016. (accessed 12-19-2016).
- [71] “Apache Hadoop.” [https://en.wikipedia.org/wiki/Apache\\_Hadoop](https://en.wikipedia.org/wiki/Apache_Hadoop), 2016. (accessed 12-19-2016).
- [72] Apache, “MapReduce Tutorial.” [https://hadoop.apache.org/docs/r1.2.1/mapred\\_tutorial.html](https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html), 2016. (accessed 12-19-2016).
- [73] S. Ghemawat, H. Gobioff, and S.-T. Leung, “The Google file system,” in *ACM SIGOPS operating systems review*, vol. 37, pp. 29–43, ACM, 2003.
- [74] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.
- [75] D. Rosenberg, “Hadoop breaks data-sorting world records.” <https://www.cnet.com/news/hadoop-breaks-data-sorting-world-records/>, 2009. (accessed 12-19-2016).
- [76] Apache, “Hadoop: Setting up a Single Node Cluster..” <https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/SingleCluster.html>, 2016. (accessed 12-19-2016).
- [77] Hortonworks, “Learn Hadoop: The Essentials Series Hortonworks 15 videos.” [https://www.youtube.com/watch?v=6UtD53BzDNk&list=PL2y\\_WpKCCNqELC4reyP-RaBqfH5QML000](https://www.youtube.com/watch?v=6UtD53BzDNk&list=PL2y_WpKCCNqELC4reyP-RaBqfH5QML000), 2016. (accessed 12-19-2016).
- [78] “Apache Hadoop HDFS.” <http://hortonworks.com/apache/hdfs/>, 2016. (accessed 12-19-2016).
- [79] “HDFS Architecture Guide.” <https://www-01.ibm.com/software/data/infosphere/hadoop/avro/>, 2013. (accessed 12-19-2016).
- [80] “C API libhdfs.” <https://hadoop.apache.org/docs/r1.2.1/libhdfs.html>, 2013. (accessed 12-19-2016).
- [81] Edureka! <https://www.youtube.com/watch?v=A02SRdyoshM>, 2016. (accessed 12-19-2016).
- [82] “Bigtable.” <https://en.wikipedia.org/wiki/Bigtable>, 2016. (accessed 12-19-2016).
- [83] “Ganglia Monitoring System.” <http://ganglia.info/>, 2016.
- [84] “What is HBase?” <https://www-01.ibm.com/software/data/infosphere/hadoop/hbase/>, 2016. (accessed 12-19-2016).

- [85] Apache, “Column Family.” <http://hbase.apache.org/0.94/book/columnfamily.html>, 2016. (accessed 12-19-2016).
- [86] “HBase - Installation.” [https://www.tutorialspoint.com/hbase/hbase\\_installation.htm](https://www.tutorialspoint.com/hbase/hbase_installation.htm), 2016. (accessed 12-19-2016).
- [87] Edureka!, “HBase Tutorial — Apache HBase Tutorial for Beginners — NoSQL Databases — Hadoop Tutorial — Edureka.” <https://www.youtube.com/watch?v=NOX6-nDtrFQ>, 2015. (accessed 12-19-2016).
- [88] “Sqoop Quick Guide.” [https://www.tutorialspoint.com/sqoop/sqoop\\_quick\\_guide.htm](https://www.tutorialspoint.com/sqoop/sqoop_quick_guide.htm), 2016. (accessed 12-19-2016).
- [89] Edureka, “Apache Pig Tutorial 1 — Understanding Pig Latin — Pig Latin Explained — Hadoop Tutorial.” <https://www.youtube.com/watch?v=Yw4hcSR-DGU>, 2016. (accessed 12-19-2016).
- [90] Sqoop. [https://www.tutorialspoint.com/sqoop/sqoop\\_installation.htm](https://www.tutorialspoint.com/sqoop/sqoop_installation.htm), 2016. (accessed 12-19-2016).
- [91] Edureka!, “Introduction to Sqoop.” <https://www.youtube.com/watch?v=UDWriTDSclo>, 2016. (accessed 12-19-2016).
- [92] “Welcome to Apache Pig!” <https://pig.apache.org/>, 2016. (accessed 12-19-2016).
- [93] Tutorialspoint, “Apache Pig - Installation.” [https://www.tutorialspoint.com/apache\\_pig/apache\\_pig\\_installation.htm/](https://www.tutorialspoint.com/apache_pig/apache_pig_installation.htm/), 2016. (accessed 12-19-2016).
- [94] Hortonworks, “Hadoop Tutorial: Apache Pig.” <https://www.youtube.com/watch?v=PQb9I-8986s>, 2016. (accessed 12-19-2016).
- [95] “AvroStorage - Load and Store Avro Data in Pig Scripts.” <https://cwiki.apache.org/confluence/display/PIG/AvroStorage>, 2016. (accessed 12-19-2016).