Real-time Pictographic Representation of the Semantic Web

Dissertation Report

Submitted by:
Ronny George Mathew,
MSc. Computer Systems Management, 2012

Guided by:
Talal Shaikh
i. Declaration

1, Ronny George Mathew, confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is also included.

Signed: ..............................................................................................

Date: .................................................................................................
ii. Abstract

The Internet holds vast amounts of valuable information. The explosion of social data on Internet has triggered users to express their opinions and feelings on the Internet. This data, when processed efficiently will yield a lot of significant information, which will help us gain new knowledge. In this research, we will be trying to extract this information and analyze it in real-time using some of the latest technologies in this field. Initially, we will evaluate some of the new technologies which we can use and finally select an appropriate method for our purpose of real-time analysis. Then, with the chosen method, extract the emotions or sentiments involved in the data. All the processed data will then need to be displayed to the end user in an easy to understand and simplistic way, at the same time extract the maximum amount of information from the data.
iii. Acknowledgement

First of all, I want to thank my supervisor, Mr. Talal Shaikh for all his support and guidance throughout this project. I also want to thank Mr. Stephen Gill, who guided us through the various stages of development of a project. I want to thank my parents and all my friends who have provided their valuable feedback to enhance the functionality of this application. Lastly, I want to thank The Almighty God for helping me complete this project successfully.
# iv. Table of Contents

i. Declaration ................................................................. 1  
ii. Abstract ........................................................................ 2  
iii. Acknowledgement ...................................................... 3  
iv. Table of Contents ...................................................... 4  
v. Table of Figures ........................................................... 7  
1. Introduction .................................................................... 8  
2. Literature review ........................................................... 10  
   2.1. Big Data .................................................................... 10  
   2.2. Extraction & Loading .................................................. 12  
      2.2.1. Flume .................................................................. 12  
      2.2.2. Crawlers ............................................................. 13  
      2.2.3. Apache Nutch ...................................................... 13  
      2.2.4. Twitter API .......................................................... 14  
      2.2.5. Facebook APIs ..................................................... 15  
      2.2.6. RSS ................................................................... 17  
      2.2.7. Evaluation Criteria .............................................. 18  
   2.3. NoSQL Databases ..................................................... 19  
      2.3.1. Overview ............................................................ 19  
      2.3.2. The Basic Principle Of NoSQL ............................. 20  
      2.3.3. Types of NoSQL Data Stores .............................. 21  
      2.3.4. SQL or NoSQL .................................................... 24  
      2.3.5. Evaluation Criteria .............................................. 24  
   2.4. Data Processing System .............................................. 25  
      2.4.1. Hadoop .............................................................. 25  
      2.4.2. Storm ................................................................. 30  
      2.4.3. S4 ...................................................................... 31  
      2.4.4. Node.js ............................................................. 32  
      2.4.5. Evaluation Criteria .............................................. 33  
   2.5. Querying Data from NoSQL ....................................... 34  
      2.5.1. Apache Pig ........................................................ 34  
      2.5.2. Apache Hive ....................................................... 35  
      2.5.3. Mongoose for MongoDB ..................................... 35
2.5.4. Evaluation Criteria .......................................................... 36
2.6. Knowledge Discovery ......................................................... 37
2.7. Sentiment Analysis .............................................................. 38
2.8. Visualizing the Data ............................................................ 40
    2.8.1. Processing.js .......................................................... 40
    2.8.2. Google Charts API .................................................. 40
    2.8.3. Google Maps API ................................................... 40
    2.8.4. AJAX Graphs ........................................................ 40
    2.8.5. Evaluation Criteria .................................................. 40
2.9. Methodology ........................................................................ 41
2.10. Additional Research ......................................................... 42
    2.10.1. PHP ......................................................................... 42
    2.10.2. HTML5 ..................................................................... 42
    2.10.3. CSS3 ....................................................................... 43
    2.10.4. JQuery ...................................................................... 43
    2.10.5. Meteor.js ............................................................... 43
    2.10.6. Express Framework .................................................. 43
    2.10.7. Socket.io ................................................................. 44
    2.10.8. Open Weather Maps API ......................................... 44
    2.10.9. Backbone.js ........................................................... 44
    2.10.10. Python ................................................................. 44
    2.10.11. R ........................................................................... 44
    2.10.12. Bootstrap ............................................................. 44
    2.10.13. AWS ..................................................................... 45
3. Requirements Analysis ............................................................ 46
    3.1. Objectives ...................................................................... 46
    3.2. Mandatory Requirements ............................................... 47
    3.3. Optional Requirements .................................................. 49
4. Design ..................................................................................... 50
5. Professional, Legal, Ethical, and Social Issues .................................. 51
6. Project Plan ............................................................................. 52
7. UI Design ............................................................................... 54
    7.1. Map View ...................................................................... 54
    7.2. Infographic View .......................................................... 55
v. Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The three V’s of Big Data (Wu X., 2012)</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>The Big Data Process (HortonWorks, 2012)</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>Flume Data Model (Apache Flume)</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>Twitter REST API (Twitter)</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>Twitter Streaming API (Twitter)</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>The CAP Diagram (Wu X., 2012)</td>
<td>19</td>
</tr>
<tr>
<td>7</td>
<td>HDFS Architecture (Humbetov S., 2012)</td>
<td>25</td>
</tr>
<tr>
<td>8</td>
<td>HDFS Packet Structure (Xu W., 2012)</td>
<td>27</td>
</tr>
<tr>
<td>9</td>
<td>The MapReduce Model (Xu W., 2012)</td>
<td>28</td>
</tr>
<tr>
<td>10</td>
<td>HDFS Job Tracker (Humbetov S., 2012)</td>
<td>27</td>
</tr>
<tr>
<td>11</td>
<td>MapReduce Example (Humbetov S., 2012)</td>
<td>29</td>
</tr>
<tr>
<td>12</td>
<td>Conceptual architecture of a trivial Storm topology (Jones T., 2012)</td>
<td>31</td>
</tr>
<tr>
<td>13</td>
<td>Simple Storm topology for the MapReduce function (Jones T., 2012)</td>
<td>31</td>
</tr>
<tr>
<td>14</td>
<td>The Node.js Event Model (Aaron S., 2011)</td>
<td>31</td>
</tr>
<tr>
<td>15</td>
<td>Typical Data Access Patterns (Wikipedia)</td>
<td>36</td>
</tr>
<tr>
<td>16</td>
<td>Analytics Data Access Pattern (Wikipedia)</td>
<td>38</td>
</tr>
<tr>
<td>17</td>
<td>Categorization of Emotions (Kalwar S., 2011)</td>
<td>39</td>
</tr>
<tr>
<td>18</td>
<td>Agile Development Methodology (Wikipedia)</td>
<td>39</td>
</tr>
<tr>
<td>19</td>
<td>The Project Plan</td>
<td>53</td>
</tr>
<tr>
<td>20</td>
<td>the Map View Layout</td>
<td>54</td>
</tr>
<tr>
<td>21</td>
<td>Designing the Marker</td>
<td>54</td>
</tr>
<tr>
<td>22</td>
<td>Designing the InfoWindow</td>
<td>55</td>
</tr>
<tr>
<td>23</td>
<td>The Map View</td>
<td>70</td>
</tr>
<tr>
<td>24</td>
<td>The InfoWindow</td>
<td>71</td>
</tr>
<tr>
<td>25</td>
<td>The Search Facility</td>
<td>71</td>
</tr>
<tr>
<td>26</td>
<td>Live Tweets Nearby</td>
<td>71</td>
</tr>
<tr>
<td>27</td>
<td>Swear Word Analysis</td>
<td>72</td>
</tr>
<tr>
<td>28</td>
<td>Mapping Weather to Sentiments</td>
<td>72</td>
</tr>
<tr>
<td>29</td>
<td>Source Analytics</td>
<td>72</td>
</tr>
<tr>
<td>30</td>
<td>Hash Tags &amp; Languages</td>
<td>73</td>
</tr>
<tr>
<td>31</td>
<td>Listing the RSS Feeds</td>
<td>73</td>
</tr>
<tr>
<td>32</td>
<td>Mandatory Requirement Checklist</td>
<td>75</td>
</tr>
<tr>
<td>33</td>
<td>Optional Requirement Checklist</td>
<td>75</td>
</tr>
<tr>
<td>34</td>
<td>Combining RSS Feeds in RSS Mix</td>
<td>92</td>
</tr>
<tr>
<td>35</td>
<td>Result of RSS Mix</td>
<td>92</td>
</tr>
</tbody>
</table>
1. Introduction

The aim of this project is to collect information from online sources and to process this information in real-time. Then analyze this data for emotions or feelings. This will help us to roughly get the current sentiment of the Internet. This information has to be processed in real-time and the result should be displayed on a web page in a simple and elegant manner. This will further help us to perform more analytics on the data and find various other observations and derive some conclusions.

The Internet is a massive reservoir of different kinds of data ranging from simple social data, photos and videos to GPS coordinates, banking transactions, weather, etc. The volume of this digital data is so large, that we will need specialized management tools and applications to analyze all of it. This huge amount of data is called Big Data (Singh S. & Singh N., 2012:1). It is so called because of the simple observation that 2.5 exabytes of data is created around the world every single day. Even the twitter feed the developers have access to, is said to be only 1% of the entire tweets (Fire Hose) happening around the world. All of this information as it is, is useless. There is also no point of storing all of this data if we can’t process and analyze it properly. The way we use it, makes all the difference.

One of the practical applications of big data analysis is that the purchase history of a customer can be stored by an online store so that it can provide tailor-made promotions for them. This can be seen nowadays when you might feel like one of the ads or ads of a certain product keeps “following” you on every web page you visit. It can also be used to analyze consumer habits, for example, Microsoft can search in real-time how many Surface devices it is selling and see what the consumers are talking about it on the web. This enables them to take immediate actions in case of an issue.

The process of evaluating and operating on all this data to find patterns, hidden relations between them and other important information is called Big Data Analytics (Singh S. & Singh N., 2012:1). This information can help a business to gain competitive advantage over another and also increase the growth, revenue and productivity. This can be achieved by analyzing the clickstreams, web server logs, call logs, social data and other
information. Big Data Analytics needs software, which makes use of advanced analytics like predictive analytics and data mining.

Most of this data is unstructured along with some structured data such as transactions. But this unstructured data cannot fit in the traditional data warehouse nor can they accommodate the additional processing overload imposed by this data. This new requirement gave birth to a new breed to Big Data Technologies. The list of the names everyone interested in Big Data Analytics has to be familiar with includes NoSQL databases, Hadoop, HDFS and MapReduce. All these technologies are part of an open-source distributed processing framework that supports Big Data.

A lot of digital data is being created every day, it has a huge potential to be of value for many purposes if collected and processed effectively. Data warehouses and Analytics used to be limited to big enterprises, but today, organizations and researchers of different industries are seeing massive potential value in warehousing the increasing wealth of digital information.

This web application can also be accessed at http://aws.ronlabz.com:8081/
2. Literature review

2.1. Big Data

Big data simply refers to the scale of data that cannot be handled by traditional methods of data warehousing. This has in turn lead to the development of new architectures and applications that can process this new form of data. These systems are highly scalable and processes data efficiently in a distributed environment.

According to the article from HortonWorks (2012:2), “Big data is a very popular term in today’s world, because of the increasing amount of information being stored and processed by today’s businesses. Today the IT notion of Big has reached the petabyte range for traditional data warehouses and is moving to exabytes range.”

But volume is only one of the several properties defining Big Data. These properties, coined by Analysts as “The Four V’s”, are as follows:

- **Volume**: The mere size of data can range from terabytes to petabytes. IBM estimates that 2.5 quintillion bytes of data are created every day. There has also been reports that 90% of all the data in the world now was created in just the last 2 years. The cyber space is estimated at 1.8 trillion GBs in size and saved in 500 quadrillion files. This size is more than doubled every 2 years. If we compare the digital universe with our physical universe, we might find that there are nearly as many bits of information as there are stars in our universe. (Singh S. & Singh N., 2012:1)

- **Velocity**: The speed of processing the data. Data needs to be collected and stored at exceptional speeds. The challenge is not only to store and manage huge amounts of data but also to extract useful information from it. (Singh S. & Singh N., 2012:1)

- **Variety**: We will need to store different kinds of data from multiple sources. There are different approaches to collecting, storing and analyzing data. The data can include structured data as well as unstructured. Most of the information used in Big Data Analytics are unstructured. Some examples of unstructured data are images, sensors, video, logs, emails, etc. (Singh S. & Singh N., 2012:1)

- **Variability**: This is the randomness of the data in terms of its meaning or context. (Singh S. & Singh N., 2012:1)

All the techniques share the same characteristics of scale, elasticity and high availability. Managing the big data can be difficult related to capturing of data, storage, search, processing, etc.
HortonWorks (2012:2) states that obtaining valuable information from big data involves transforming raw data into usable information will offers new insights to businesses, which in turn helps to make better business decisions.

Figure 2: The Big Data Process (HortonWorks, 2012)

**Input:** This stage involved the collection of various kinds of data from multiple sources. The raw input data can be unstructured like logs, sensor values, social activity, emails, etc. also structured data like database records, transactions, etc.

**Output:** Output from the processing system can be displayed directly to end users in form of reports, or visual tools. It can also be loaded into relational databases or other systems for further processing.

**Extraction & Loading:** This step involves the processes of extracting and loading the input data into the processing systems.

**Storage:** The processing system must provide an efficient way of storing information. Apache Hadoop employs HDFS, which helps to distribute the storage of important data across a cluster of systems. Hadoop enables batch processing of massive collection of data.

**Processing:** The system must also efficiently process the data from the input stage and produce meaningful information from it. For example, Apache Hadoop employs MapReduce for this purpose. Hadoop also preserves the original data, so that we can do further processing even at a later time.
2.2. Extraction & Loading

In this section, we will discuss some of the methods to extract and load information into our processing system.

2.2.1. Flume
Apache Flume is a distributed and reliable service for collecting, aggregating and moving large amounts of log data. Its main goal is to deliver data from multiple applications or web servers to Hadoop’s HDFS. It has a simple and flexible architecture based on streaming data. It is robust and fault-tolerant with tunable reliability mechanisms and many failover and recovery mechanisms. It uses a simple extensible data model that allows for online analytic applications. Flume supports a number of mechanisms to ingest data from external sources (Apache Flume, 2012).

A Flume event is defined as a unit of data flow having a byte payload and optional set of string attributes. A Flume agent is a (JVM) process that hosts the components through which events flow from an external source to the next destination (hop).

A Flume source consumes events delivered to it by an external source like a web server. The external source sends events to Flume in a format that is recognized by the target Flume source. When a Flume source receives an event, it stores it into one or more channels. The channel is a passive store that keeps the event until it’s consumed by a Flume sink. The file channel is one example – it is backed by the local file system. The sink removes the event from the channel and puts it into an external repository like HDFS (via Flume HDFS sink) or forwards it to the Flume source of the next Flume agent (next hop) in the flow. The source and sink within the given agent run asynchronously with the events staged in the channel.

Figure 3: Flume Data Model (Apache Flume)

A Flume source consumes events delivered to it by an external source like a web server. The external source sends events to Flume in a format that is recognized by the target Flume source. When a Flume source receives an event, it stores it into one or more channels. The channel is a passive store that keeps the event until it’s consumed by a Flume sink. The file channel is one example – it is backed by the local file system. The sink removes the event from the channel and puts it into an external repository like HDFS (via Flume HDFS sink) or forwards it to the Flume source of the next Flume agent (next hop) in the flow. The source and sink within the given agent run asynchronously with the events staged in the channel.
2.2.2. Crawlers
A crawler scans the contents of websites and feeds this data into the database or a data store. There are various open source scripts available for crawlers in different languages such as Java, C#, etc. Crawler4j is an open source crawler based on java that provides an easy to use interface for web crawling. Crowl is another crawler that is designed specifically for Hadoop. Crowl can crawl through a single feed URL or even a collection of feed URLs. Crowl gives HTML free content of the feed entry along with the image URLs and stores the image thumbnails in feeds to a local directory. Scrapy is another open-source alternative that is simple and fast framework for web crawlers written in Python. Node.io is a website scraping tool to obtain contents in real-time and can be used in conjunction with node.js.

2.2.3. Apache Nutch
Apache Nutch is an open-source Java crawler. It finds webpage hyperlinks in an automated manner, which reduces a lot of maintenance works. Eg: checking broken links. Apache Solr is an open source full-text search framework, with Solr we can search pages from Nutch. Apache Nutch natively supports Solr. (Apache Nutch, 2011)

Apache Nutch is actually a package of Web Crawler with Hadoop and Lucene (For indexing). Key features of Nutch are:

• Seeds and Crawl Filters.
• Crawl Depths.
• Fetch Lists and Partitioning.
• Segments.
• Indexing using Lucene.

Apache Nutch is worth a mention because it is the complete package including Hadoop and the crawler needed for data processing. Since, it is compatible with Solr and Lucene, we can use it if we are planning on adding any search capabilities to our system.
2.2.4. Twitter API

Twitter provides a set of streaming APIs for developers that provides low latency access to a sample of twitter’s global stream of data. All the messages will be transferred to the streaming client as they occur without any overhead of REST endpoint polling.

According to Twitter, the different kinds of streaming endpoints are,

1. **Public Streams**: Stream providing all the public data flowing through Twitter. This is used for all data mining applications and also to follow specific users or topics. It offers sample of public data flowing through Twitter. Once a connection is established, a stream of tweets are delivered without any need for any polling.

2. **User Streams**: Single user streams provide the data for a single user’s view of Twitter. Data and events specific for a particular user are provided through this stream.

3. **Site Streams**: Site streams are the multi-user version of user streams intended for servers that needs to connect to Twitter on behalf of multiple users. It allows services such as mobile push or web sites, to get real-time updates. Events can be streamed for users who grant OAuth access for the application.

Tweets are basic building blocks of Twitter. User tweets are like Status updates, they can be embedded, replied to, make/undo favorite and even deleted. Users can broadcast tweets of others by retweeting (Twitter).

As per Twitter, the streaming API requires an open HTTP connection on the contrary to REST API. The below figure shows the case of a web application that takes user requests, queries Twitter API, formats and prints result to user.

![Figure 4: Twitter REST API (Twitter)](image-url)
An application that connects with the Streaming API cannot open a connection when there is a user request as shown above but instead, it has to maintain a streaming connection that is run in a process separate from that running the HTTP request.

Figure 5: Twitter Streaming API (Twitter)

The streaming process get the tweets and process it before storing it to a data store while the HTTP process handles queries to the data store for results. Even though this method is a little more complicated, real-time streaming has its own benefits.

2.2.5. Facebook APIs

2.2.5.1. Facebook Open Stream API

Facebook’s Open Stream API allows users to use applications to interact and read their stream anywhere. Developers can also access stories that was published into stream and show them in the application on mobile, website or desktops. As per Facebook, APIs such as Stream.get, Stream.publish and FQL tables enables direct access to the stream. It enables developers’ access the streams on behalf of a user and get access to his content. We can filter, combine or display the data in any way we need. Users can also publish back into the stream, add likes and comments to any items in the stream.
2.2.5.2. Graph API

Graph API is basic way of retrieving or posting data on Facebook. With Graph API, we can subscribe to changes on all properties of the user object. We can subscribe to all the following: feeds, activities, friends, music, interests, books, movies, television, likes, check-ins, events, location (Facebook).

As per Facebook, with the Graph API we can:

- **Read Pictures**: Pictures are available for many objects and also the most commonly used objects in the graph. This does not include the albums or photographs. An example of a picture that can be access from Graph API is the profile picture of a user which can be obtained by entering the following link as the source:
  
  ```
  https://graph.facebook.com/<username>/picture?type=large
  ```

  Type can be square, small, medium or large. We can also use attributes like width and height to get the image in the required size.

- **Using Graph API**, we can also get the information of a user. This returned results can also be filtered. The following link will return the information of the user according to the fields specified.
  
  ```
  https://graph.facebook.com/ronnygeo?fields=id,name,picture
  ```

- **Pagination**: Graph API gets data in small sets. We can also page forward and backwards the returned results in time.

- **We can search public objects with Graph API**.

- **All the dates returned through Graph follows the ISO-8601 formatted strings**. For example, the `creation_time` object is as shown:

  ```
  {
    "creation_time": "2012-12-15T14:00:02+0000",
    "id": "19292868552_522414651110169"
  }
  ```

  We also have the option to override the date format by specifying `date_format` parameter. The format strings are similar to those accepted by the PHP `date` function.

- **Most of the data accessed with Graph API requires user permissions i.e. the user must grant the required access to an application to read any data beyond public data or for writing data to a user’s timeline (Facebook)**.

Graph API provides real-time updates, batch requests and field expansions. Real-time updates enable applications to subscribe to data changes and also cache data and receive updates rather than polling on a regular basis. Facebook will push the updates whenever a change occurs. This greatly improves the reliability and performance of the application. We can also process a bunch of requests together in a batch and run them all at once instead of individual queries. We can also get multiple objects in a single query. This improves the latency and overhead involved in the communication (Facebook).
2.2.5.3. FQL

FQL stands for Facebook Query Language. FQL allows us to use an interface similar to SQL to easily fetch the data exposed by Graph API. It also performs multiple query batching into a single one.

2.2.6. RSS

RSS stands for Rich Site Summary (or sometimes Really Simple Syndication) is standard format for web feed to publish the latest updates of news headlines, blogs, etc. It includes a summary and sometimes even the full text of the original article along with the metadata such as publishing dates and authorship. RSS feeds can make use of reader softwares that can be desktop-based, mobile-based or web-based. The user can add the URI of the feed to the reader to subscribe. Atom is another feed similar to RSS feeds. RSS documents are XML formatted plain text documents. It is easy to read by humans as well as systems. An example structure of an RSS feed is as shown:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0">
  <channel>
    <title>RSS Title</title>
    <description>This is an example of an RSS feed</description>
    <link>http://www.someexamplerssdomain.com/main.html</link>
    <lastBuildDate>Mon, 06 Sep 2010 00:01:00 +0000</lastBuildDate>
    <pubDate>Mon, 06 Sep 2009 16:20:00 +0000</pubDate>
    <ttl>1800</ttl>
    <item>
      <title>Example entry</title>
      <description>Here is some text containing an interesting description.</description>
      <link>http://www.wikipedia.org/</link>
      <guid>unique string per item</guid>
      <pubDate>Mon, 06 Sep 2009 16:20:00 +0000</pubDate>
    </item>
  </channel>
</rss>
```
2.2.7. Evaluation Criteria
In this section, we will evaluate the technologies mentioned and choose the appropriate one for our application.

Since, our application needs to process data in real time, the data source also should be able to emit data in real-time. For this purpose, we will use the Twitter Streaming API as our main data source. The advantage of Twitter Streaming API is that it provides us with a sample of the public stream of tweets in real-time. We also need to setup the Streaming endpoints of Twitter so that tweets can be fed to the processing system in real-time. Twitter input will be the main (mandatory) source of information for our project. So, we will setup twitter’s streaming API to send updates to the processing system, which will in turn provide us with the real-time processed information on the web page. We will use the Public stream since it contains a sample of all the tweets happening around the world. Using Site stream is not effective since each user has to approve our application to enable streaming data.

We can make use of the Open Stream API and Graph API to load information from Facebook into our processing system. The issue with these APIs is that Facebook still does not provide anything similar to twitter’s streaming API to get all of the public feeds. One solution for this is to create an application in Facebook that gets the users’ permission to allow streaming their data to our application.

Flume, even though fully compatible with NoSQL databases, are not useful for our purpose since it requires us to setup a streaming data point on each node (web server). Flume is basically used to collect data from streaming servers.

As an optional requirement for this project, we can create a crawler that crawls through different websites, blogs, forums, etc. and collect various kinds of data. This data can be then sent for processing to the data store. We use any of the crawler libraries mentioned above that is compatible with our processing system. But since our primary purpose is real-time data analysis, crawlers cannot be used as our primary source of information since the data from crawlers might not be live, and can also be very old. Instead, we can possibly use this information as an additional input.

The next optional requirement can be the addition of RSS feeds to the application. RSS feeds provide us with the latest data from the source along with the Time stamp of the feed. We can setup our application to listen to RSS feeds from various news sites and perform some operation on it.


2.3. NoSQL Databases

2.3.1. Overview

The rapid growth of data have led to the development of new technologies and systems that gives good horizontal scalability for easy read/write database operations that can be distributed over multiple server.

Bakshi K. (2012:1) states that the traditional method of shaping data includes relational databases and data schemas. In case of structured data, we can deploy solutions like data warehouses and data marts. Data warehouse is a relational system for storing, processing and reporting of data. Data mart is a layer used to access the data stored in a Data Warehouse. The main focus of data warehousing is the data storage where the data is cleaned, processed, catalogued and is made readily available for analytics and other data mining applications. Both data warehouses and marts are based on SQL. The two main approaches of data storage in data warehouse are:

- **Dimensional**: Transaction data are partitioned into “facts” table. The table contains the reference information that gives context to the fact. (Bakshi K., 2012:1)
- **Normalized**: The tables grouped by subject, which reflects different categories such as the customer data, product data and so on. The normalized structures split data into entities that creates tables in relational database. (Bakshi K., 2012:1)

When the demand for finding and storing all the Internet data increased, searching became impossible with relational databases such as SQL.

Google was the first one to come up with a solution for this problem by creating the Google File System (GFS). It is a byte-stream based data file randomly partitioned to hundreds and thousands of nodes in a cluster. GFS was then merged with another technique called MapReduce, which enabled programmers to process big data. An apache open source implementation of Google’s solution is the popular Hadoop platform with its own HDFS storage layer. Hadoop and HDFS have grown to become the dominant platform for big data analytics (Bakshi K., 2012:1).

These new systems are called “NoSQL” data stores. NoSQL stands for “Not Only SQL”.

Cattell R. (2010) states that NoSQL systems have six main features:

1. Horizontal Scalability, i.e. the support for multiple servers.
2. Data Replication across all the servers.
3. A simple call level protocol or interface instead of SQL binding.
4. Efficient use of distributed indexes and RAM for storage of data.
5. Dynamic addition of attributes to records.
Cattell R. (2010) also states that one of the key features of these systems is the “shared nothing” horizontal scaling where data is replicated and partitioned across hundreds and thousands of servers. This allows for large number of simple read/write operations in a second. This simple operation load is called OLTP (Online Transaction Processing), but it is also common in modern web applications. NoSQL databases does not generally provide ACID properties, i.e. the updates are gradually propagated and there is only a limited guarantee on read consistency.

2.3.2. The Basic Principle Of NoSQL

The following principles form the foundations of NoSQL systems. **CAP Theorem**: CAP stands for Consistency, Availability and tolerance for network Partition. The core idea of this theorem is that a distributed system cannot meet all the three needs simultaneously, but only two of them can be met. (Wu X., 2012)

According to CAP Theorem, we have three options to design a system, CA, AP or CP. Availability is the most important for Web 2.0 sites, so when designing systems, we can use only CA and AP orientation. Also for Web 2.0 sites, consistency is of least significance in contrast to availability and partition tolerance.

NoSQL databases have more scalable architectures, which relaxes the consistency. This relaxing consistency is called Eventual Consistency. This allows writes to occur without system-wide locks. But this can lead to conflicting writes and inconsistent reads. The application should be designed in such a way that this does not happen.

**BASE Theorem**: BASE theorem is the result of CAP theorem put into practice. BASE model is totally different from ACID model. BASE stands for Basically Available, Soft-state and Eventual consistency. (Wu X., 2012)

**Eventual Consistency**: This is one of the consistency models used in parallel programming. It says that in a sufficiently long period of time, over which no changes are sent, all updates can be expected to propagate through the system eventually and all replicas will be consistent (Wang G. and Tang J., 2012: 1333).

Consistency is divided into two sides:

1. The Client Side Consistency.
2. The Server Side Consistency.

According to Wang G. and Tang J. (2012: 1333), Client side consistency deals with how and when clients see updates made to a data object in storage system. They are mainly three types:

a. **Strong Consistency**: Subsequent access after update will return the new value.

b. **Weak Consistency**: The system does not guarantee subsequent accesses will return the updated value. A lot of conditions have to be met before the value will
be returned. There is an inconsistency window between the update and the moment when a client sees the updated value.

c. **Eventual Consistency:** The eventual consistency is a specific form of weak consistency. The updates are eventually propagated to all nodes, but many of them deploy mechanisms for some consistency, such as the Multi-Version Concurrency Control (MVCC). The system guarantees that if new updates are not made to the object, eventually the updated value will be returned in all accesses. The inconsistency windows can be determined based on certain factors like communication delays and replicas involved (Wang G., 2012: 1333).

Bakshi K. (2012:1) states, “There are different approaches adopted by NoSQL for storing and managing data, but it share the goals of scalability, data model flexibility and easy application development and deployment.” NoSQL separates the concepts of data storage and data management, while traditional databases tries to accomplish both with databases. One of key concepts of NoSQL is database focuses on only data storage and provides low-level access to data management layer so that data management can be done using the application layer using the same logic of SQL or other similar languages.

NoSQL databases are also called schema-free. This design helps applications to easily change the structure of the data without recreating the entire table. This creates high flexibility in storing different kinds of data. The validity and integrity of the data is enforced at the data management layer. The process of storing data on different servers based on a key by providing horizontal scalability is called Sharding. Some of these systems also perform vertical scaling, where the parts of a single file record are stored on different servers. (Bakshi K.)

### 2.3.3. Types of NoSQL Data Stores

According to Cattell R. (2010), NoSQL systems stores sets of key-value pairs, but employs various data structures, namely:

- **Tuple:** A row in relational table, attribute names are already defined in the schema and the values must be scalar. The values are accessed with the attribute name opposed to array/list where they are accessed by ordinal position.

- **Document:** Nested documents or lists or even scalar values. The attribute names are defined dynamically for each document at runtime. The global schema does not have document attributes listed and values have a wider range.

- **Extensible Record:** A hybrid of tuple and document, where attribute families are defined in the schema, but attributes can be added for each record. The attributes can be list valued.

- **Object:** Similar to objects in a programming language but lacks the functions and methods. Values in an object can be references and nested objects.

NoSQL systems are of different kinds based on their data model. As per Cattell R. (2010), they are categorized as follows:

a. **Key Value:** These store values and the index to locate them. The programmer defines the keys and since it offers fast lookups typically used for content
caching. The stored data have no schema. Some examples of key value stores are Redis, Voldemort, Oracle BDB, etc.

b. **Document Data:** These systems store and index documents. They can be queried using a simple mechanism. Used for Web applications that needs to be tolerant of incomplete data. Some cons of these are performance and there is no standard query syntax. Some examples of this are CouchDB, MongoDB, etc.

c. **Extensible Record:** They store extensible records that can be partitioned horizontally and vertically across various nodes. Also called Wide Column Stores.

d. **Relational Databases:** Traditional databases that stores and indexes tuples that allows horizontal scaling.

### 2.3.3.1. Key-Value Stores

R. Cattell (2010: 3) says that key value stores employs a model close to the memcache distributed in-memory cache with a single key-value index for the entire data. It provides a persistent mechanism and additional functionality like replication, locking, versioning, sorting, transactions and other features. It does not offer secondary keys or indices. An advanced key-value store is Project Voldemort written in Java, which is an open source implementation with contributions from LinkedIn. It provides updates with MVCC (Multi-version concurrency control). The updates are replicated asynchronously without any surety for consistent data. Riak is another open-source key-value store written in Erlang. It is an advanced key-value store with much more features than Voldemort.

### 2.3.3.2. Document Stores

R. Cattell (2010:5) states that Document stores store data that is more complex in structure. Documents can be any type of pointer-less object. It supports secondary indices, nested documents or lists and different kinds of documents per database. SimpleDB is an Amazon’s cloud computing product that is easy to use than other document stores since it doesn’t allow nesting of documents. CouchDB is another example of document store written in Erlang. It is an apache project similar to SimpleDB but much more richer. A document can have attribute values that are Scalar (Numeric, Boolean or text) or a Compound (Lists or Document). It employs MapReduce mechanisms to process the queries in parallel over many nodes. MongoDB is an open-source GPL document written in C++ similar to CouchDB but has its own distinctive features.

### 2.3.3.3. Extensible Record Stores

The data is stored in rows and column family group rows. The column families’ table must be defined along with the table schema definition. Bakshi K. (2012:2) states, “HBase is an Apache project similar to Google’s BigTable written in Java. HBase deploys Hadoop Distributed File System (HDFS) instead of Google File System (GFS). It is a column-oriented distributed database built on HDFS.” It can hold very large or sparsely populated tables on nodes made from simple commodity hardwares. It adds the updates to the memory and then periodically writes to the disk. HBase has a distributed master-slave architecture that depends on the quorum service. The HBase master nodes handle a cluster of slave nodes. Hadoop file system APIs is used to maintain all the HBase data.
The updates are appended to the end of the file to remove seeks. These files are then periodically compacted. Recovery can also be done in case of server crashes. HBase uses MapReduce to distribute operations effectively. It has a log structure which merges file index to allow quick range queries and sorting. APIs for Java, Thrift and REST are available along with support for ODBC and JDBC.

Cassandra is highly scalable, distributed, structured key-value store with eventual consistency. Cassandra is similar to HBase in its functionality and data model. Cassandra is written in Java with Apache Licensing. It was open sourced by Facebook and now being supported by DataStax. Facebook along with a lot of other companies use Cassandra. There are column groups, cached updates in memory that are flushed to disk. The disk contents are compacted periodically. It also does partitioning and replication with fully automatic detection of failure and recovery. Cassandra adds available nodes into a cluster automatically. It detects node failure with the phi accrual algorithm and cluster memberships are determined with a gossip-style algorithm in a distributed fashion. It’s been reported that Cassandra has scaled to about 150 production machines at Facebook. For applications where eventual consistency is not enough, “quorum reads” provides a way to get the latest data. There is also support for conflict resolution and versioning.

2.3.3.4. Scalable Relational Systems
Relational DBMS are entirely different from NoSQL solutions since, they have a preset schema, ACID transactions and SQL. The area where RDBMS fails is scalability. Recent developments in MySQL have resulted in new systems that are more scalable. Products such as VoltDB and Clustrix provide good per-node performance along with scalability. Advantages of these new RDBMS to NoSQL systems are larger scope transactions and operations.

2.3.3.5. Other NoSQL Database Systems

Graph Database Systems: Efficient distributed storage and queries of a graph of nodes along with references. Good for social networking and to provide recommendations. The issue with this kind of system is that to get a clear answer we have to move along the full graph and it is hard to cluster. Some examples are Infinite Graph, Neo4j, InfoGrid and OrientDB.

Object-Oriented Database Systems: These systems create these objects as programming language objects and provides efficient distributed storage of graph of objects.

Distributed Object-oriented Stores: These are similar to Object Oriented DBMS, but they distribute the object graphs over multiple servers. It is good for applications with fast and extensive reference following while also providing ACID transactions.
2.3.4. SQL or NoSQL
SQL and NoSQL both of them have their own pros and cons. Let us just summarize some of the main features of both as per Cattell R. (2010):

a. Relational Databases provides a common interface with SQL, relational schema and transactions that is a plus when it comes to training, data interchange and continuity.

b. NoSQL systems are highly scalable, i.e. They can be distributed over multiple number of nodes.

c. In cases where simple object lookup based on a key, key value stores is adequate and is easier to understand than relational databases.

d. NoSQL provides high flexibility where the schema of a table can be altered without any modifications. This allows for each object in a collection to have different attributes.

e. Multinode, multitable operations such as join is easier on relational databases while compared to NoSQL.

f. NoSQL supports unstructured data forms along with structured data.

2.3.5. Evaluation Criteria
Since our application will basically deal with unstructured data, we will use a NoSQL databases to store and process our data. NoSQL databases also allows flexible schema so that we don’t have to define a schema for the data beforehand. We will primarily setup a NoSQL database to deploy our lexicon list for sentiments.

We will choose MongoDB for this purpose, since it has come out as the best in a lot of comparisons in terms of lookup speeds, stability and performance. Since, we need each of the word from a text to be compared with the lexicon list, the lookup needs to happen at a really fast pace. MongoDB also has a vibrant and active community.

As an optional requirement, we can also setup an Extensible key store like HBase (Hadoop) or Cassandra for storing our data for further processing. It will also aid us in storing a variety of contents that can be more than the usual key-value pair. This will also enhance and extend the functionality of this application to new horizons.
2.4. Data Processing System

The data processing system we will use need to perform various operations on the data in real-time. In this section, we will discuss various new technologies that can be used for our processing needs.

2.4.1. Hadoop

Apache Hadoop is unique in its ability to simultaneously process and analyze complex and disparate types of data. MapReduce, in conjunction with the Hadoop Distributed File System (HDFS) and HBase database, which forms part of the Apache Hadoop project, is a modern approach to analyze unstructured data.

Apache Hadoop is a data processing platform designed and built from scratch to process huge amounts of data. Hadoop got its start from Nutch, which we described previously. Its power lies in its ability to scale even by using commodity hardware. Thereby, reducing the infrastructure setup costs and helps to focus on the important part. It is a distributed file system that runs on clusters ranging from a single computer to many thousands. Hadoop was inspired by two Google projects, MapReduce and Google File System (GFS).

Humbetov S. (2012) states that Hadoop can run MapReduce programs coded in different languages like Java, C++, Python and Ruby. MapReduce jobs are parallel, so data processing can be done on a large scale with the required number of machines. Today, there are Hadoop clusters that store petabytes of data. The basic concept is to split up large inputs of data into smaller blocks so that they can be individually processed on different machines. This way we can prevent the bottleneck during data transfer between the nodes to obtain better performance. In the end, the user does not have to worry about machine failure, data availability and coordination.

According to Borkar V. et al (2012:46), some of the advantages of using Hadoop are as follows:

1. Open source against costly software licenses.
2. Multiple layers and components to manage data against those having only top level query API.
3. Support for file-based external data access against convert the data to table schemas.
4. Automatic and incremental job recovery against rolling back long jobs to their start and beginning all over again on failure.
5. Automatic data storage and placement when data grows and machines come and go against manual DBA-driven placement.
6. Replication and Machine Fail-Over without operator intervention against manual DBA driven recovery activities.

According to Borkar V. et al (2012:46), Some of the cons of this are as follows:

1. Creates an impedance mismatch because of very large byte sequential file abstraction.
2. Using high-level language layers to manage MapReduce jobs can lead to suboptimal performances.
3. Since, the data placement is entirely random, the only option to query the data is to query all of the data files.
4. The flexible schema-less model means information about the structure of the table is only known to the applications using the model. This can create application maintenance problems.
5. Coupling the front and back end of big data platforms can be difficult to setup.
6. The computational model of Hadoop is heavy because of always sorting data flowing between map and reduce, always writing temporary data to HDFS between jobs in a multi job query plan, etc.

MapReduce along with HDFS defines the core system of Hadoop. MapReduce provides the computational model while HDFS handles the distributed storage. MapReduce and HDFS are designed to work together. While MapReduce takes care of the processing, HDFS provides high throughput of data.

2.4.1.1. HDFS
Hadoop Distributed File System (HDFS) is the open source implementation of a distributed file system that can hold huge amounts of data (in the range of terabytes or petabytes) and provide high throughput access to this information. It is the core component of Hadoop and saves all the data needed for MapReduce jobs. The files are stored across many machines in a redundant fashion to ensure their availability and fault-tolerance to parallel applications.

HDFS uses a Master and Slave Architecture that isolates the data from the namespace. Each cluster consists of a single NameNode, which is the master server that manages the namespace that contains the metadata of the file system and regulates file access to clients. The NameNode is the directory namespace manager for HDFS. There are also a number of DataNodes, which is used to store the data in the cluster. A DataNode stores a series of named blocks. Xu W. et al (2012:1059) states “decoupling of data and namespace helps in increasing the scalability of disk operations in HDFS as metadata will be quickly processed by NameNode and the data will be distributed over the DataNodes.” A single Hadoop cluster typically consists of a lot of DataNodes. The NameNode performs the namespace operations like opening, closing and renaming files and directories. It is also responsible for the mapping of blocks into the DataNodes. The DataNodes serves read and write requests of the clients. They also perform creation, deletion and replication of blocks depending on the commands from the NameNode.
However, this structure can create a potential bottleneck as all the data needs to be processed by the NameNode initially. When a client asks for a particular block, the NameNode responds with the block ID and the corresponding DataNode’s pipelines. The client application can then read, write or even delete blocks. It can also copy blocks to/from other DataNodes.

The NameNode manages block replication. If a node is unresponsive for a certain time, the NameNode replicates the blocks that were on that node to another node to maintain the replication level. The NameNode is typically not used as a DataNode, except for small clusters. DataNodes also run a TaskTracker to obtain MapReduce jobs from the JobTracker. The JobTracker runs on the NameNode and is responsible for accepting jobs submitted by users. The JobTracker assigns MapReduce jobs to TaskTrackers on DataNodes, monitors the tasks and take required actions in case of node failure.

In HDFS, a file is split into blocks and these blocks are saved in the DataNodes. As per Xu W. (2012:1059), A basic HDFS block is 64MB in size. The blocks will be replicated to other DataNodes to ensure data redundancy over hardware failure and corrupted blocks. The block data transfer occurs in packets. Each packet consists of multiple chunks and header which contains the packet length, block offset, sequence number, if it’s the last packet or not, actual data length. A packet is 64KB in size by default. A chunk contains the original data and associated checksum. The chunk contains 512 bytes of original data and 4 bytes of checksum. The typical structure of a packet is as shown below:

![HDFS Packet Structure](Xu W., 2012)

2.4.1.2. MapReduce

MapReduce provides a mechanism for programmers to leverage the distributed systems for processing data sets. It is a big data processing framework, which is rapidly becoming the standard in both industry and academia because of its simplicity, scalability and failover properties.

According to Humbetov S. (2012), MapReduce jobs are run daily on Google clusters, processing more than twenty petabytes of data every day. The idea of MapReduce is to hide the complex details of parallelization, data distribution and fault-tolerance. In addition to computational problems, we need to define the parameters for controlling parallelism and data distribution. Hadoop is similar to Google MapReduce and uses multiple machines to distribute data processing.
MapReduce job consists of two distinct phases:

1. **Map Phase**: This phase divides the data into smaller blocks and assigns it to Mapper for processing. The input to the Mapper is a set of Key-value pairs \((k, v)\) and the map function is called on each of them. The output of mapper is also a sorted list of (key, value) pairs. This list is then grouped and passed (also called shuffling) to the next phase.

2. **Reduce Phase**: This phase combines the input and processes it to produces an output. The Reduce phase processes these intermediate keys from the Mapper, groups them and produces zero or aggregated results. This output is written to HDFS.

In MapReduce, the users have to define only the Map and Reduce function, the framework handles the parallelization and failover hiding all the messy details of parallelization, fault-tolerance, data distribution and load balancing through the process. Hadoop uses a distributed file system to read and write data. MapReduce uses HDFS, so the I/O performance of MapReduce depends on HDFS. An overview of the MapReduce workflow is as shown.

![Figure 9: The MapReduce Model (Xu W., 2012)](image_url)

During the Map phase, associated Map function is performed on the key/value pair of input and creates intermediate key/value pairs. During the shuffle phase these intermediate key/value pairs are sorted and grouped by the key. The shuffling phase is entirely handled by the framework. Users can specify the parameters for sorting and grouping of data. This will create a set of key/value pairs where each pair has all the values corresponding to that key. These are then partitioned and grouped by key and then passed to the Reduce phase. The reduce phase create a zero or one output value. The Map function is designed in such a way that multiple instances of the map function can be run simultaneously, so the work can be distributed. This is where the program divides the tasks.
When a MapReduce job is run on Hadoop, the job is first sent to the NameNode where the JobTracker is located. The JobTracker has multiple “slave” nodes or TaskTrackers. The TaskTrackers report to the JobTracker and asks for new work when they are idle. The JobTracker divides the map tasks between the TaskTrackers with this process, so they all process the data simultaneously in parallel. The JobTracker also keeps track of the status of each tracker. In cases where a tracker fails, the tasks are redistributed to another tracker, causing only a small delay. If any particular node is running slow, no new jobs are passed it after finishing the current job.

Consider the following scenario, where we have two files. These two files are passed onto the Mapper. The mapper creates intermediate key value pairs as shown. These are then sorted and then grouped by the key into a list. This sorted list is then passed to the Reduce function, which will create the final output.

![HDFS Job Tracker](image)

Figure 10: HDFS Job Tracker (Humbetov S., 2012)

Even though Hadoop and HDFS will be an optional requirement in our project, the underlying concepts of MapReduce and Task Trackers, etc. remains the same for most of the NoSQL systems and was worth a mention.

![MapReduce Example](image)

Figure 11: MapReduce Example (Humbetov S., 2012)
2.4.2. Storm

Storm is one such system developed by Nathan Marz from Twitter in clojure. Storm works on the streaming data and does continuous processing on it. Storm is a complex event-processing system (CEP Systems). CEP systems are of two kinds, computation and detection-oriented, each of which can be done with Storm through user-defined algorithms. CEP’s can be used to find meaningful events from a flood of events, and take actions on those events in real-time.

Jones T (2012:2) states “Storm is an open-source, real-time data processing system that differs from others since it’s intended for distributed real-time processing and is not dependent on any language.” Storm is just a computation system and does not include any storage. Storm can work with data coming from dynamic sources and even on data from other databases in this way. It supports topologies where unterminated data streams can be transformed. These transformations never stop and are continuously processing data as it arrives.

We can see one application of Storm in the generation of trend information in Twitter. Twitter intercepts emerging trends from millions of tweets and maintains it at a national and local level. This helps twitter to create real-time trending topics. This is implemented in Storm by continuously analyzing the twitter data.

Storm was developed in Clojure, which is a modern version of the Lisp language. It is a functional style virtual machine based language that runs on Java Virtual Machine but we can write applications for Storm in any language using the Thrift adapter. Adapters are available for PHP, Perl, JRuby and Scala. An SQL adapter is also available for Storm that allows data streaming into the topology.

Jones T. (2012: 2) points out that storm has an interesting features when it comes to fault tolerance and management. Storm uses guaranteed message processing where each tuple is processed fully throughout the topology. If a tuple is not processed, it is automatically replayed from the spout. Storm also performs fault detection at the task level, where on task failure, messages are quickly reassigned to restart processing. Storm has more intelligent process management than Hadoop. ZeroMQ maximized messaging performance along with reliability and add congestion detection.

Key Components of Storm

According to Jones T. (2012: 3), Storm has a data flow model where data flows continuously through a network of transformation entities. The data flow abstraction is called a Stream, which is an unbounded set of tuples. The tuple is a structure that represents standard data types (integers, floats and byte-arrays) or user-defined types with additional serialization code. A unique ID is assigned for each stream, which can be used to create topologies of data sources and sinks. Spouts are the source of streams, which gets the data from external sources into Storm.
The sinks (entities which makes the transformations) are called Bolts. It performs a single operation on a stream. They perform all the processing in the Storm topology. Bolts can perform traditional MapReduce functions or more complex functions like aggregation, filtering or communication with external entities like a database. Storm topology implements multiple transformations and therefore, requires multiple bolts with independent tuple streams. Spouts and bolts are both setup as tasks in a Linux System.

Storm can be easily setup to do MapReduce for word frequency. As shown in Figure 13, a spout creates the data stream and a bolt performs the Map function (to separate the words in the stream). The output stream of the map bolt will then flow into another bolt which will do the Reduce function (to Aggregate words into counts).

Bolts can stream data to multiple bolts as well as receive data from multiple sources. Storm uses the concept of stream groupings, which implements shuffling (random but equal distribution of tuples to bolts) or field grouping (stream partitioning based upon fields of the stream).

One of the most fascinating features of Storm is the concept of guaranteed message processing. Storm can guarantee that every tuple that is emitted from a spout will be processed. If it’s not, Storm replays the tuple from the spout.

2.4.3. S4

S4 is another real-time computation system, which is a general-purpose, distributed, scalable, fault-tolerant, pluggable platform. It allows programmers to develop applications for processing continuous streams of data easily. Yahoo! initially developed S4 and it is now an Apache incubator project that is under development.
2.4.4. Node.js

Node.js is a platform built on chrome’s JavaScript runtime for building scalable and fast network applications easily. As per Manuel K. (2012), “Node.js use non-blocking I/O, event-driven model, which makes it efficient and lightweight for real-time data intensive applications across distributed systems.” Node.js can help to keep a connection open for streaming data. The goal of Node.js is to provide simple way to create scalable network programs. It can handle multiple client connections concurrently and tells the operating system to be notified when a new connection is made and then to go to sleep. If there is a new connection, it executes a callback. An example of a simple Node Web server is as follows:

```javascript
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World
');
}).listen(1337, "127.0.0.1");
console.log('Server running at http://127.0.0.1:1337/');
```

With node.js, we need to setup the HTTP server, the router which maps requests to the appropriate request handlers, the request handlers and the upload handlers. The beauty lies in the fact that everything from the server to the request handlers are under our control and we can customize it to any extend that we need. Node.js works on an event-driven model with asynchronous callbacks by making use of an event loop. The synchronous callbacks of node.js help to prevent locking of the event loop. For example,

```javascript
database.query("SELECT * FROM hugetable", function(rows) { var result = rows; });
console.log("Hello World");
```

The database query might take some time to execute and lock the event loop. In the above example, node.js handles this request in an asynchronous fashion. The function database.query( ) should be defined as an asynchronous function. Here, node does not wait for the query to return results, it simply makes note of the function to execute when the results are returned and moves forward with other tasks. In this case, it will print “Hello World” immediately after executing the query function. When the results arrive, the callback function is called. Node cycles through the event loop continuously even when there is nothing to do, waiting for events such as the return of results from the previous query.

![Node.js Processing Model](image)

*Figure 14: The Node.js Event Model (Aaron S., 2011)*
Node does better memory management under heavy loads. Since node does not perform I/O operations directly, it never blocks the process. Thereby preventing deadlocks on the process and helping developers to develop fast systems. Since, node makes use of events it uses only a single thread for operations. This can be a drawback in case of multicore machines. But various modules have been developed for this purpose. Cluster.js module for node provides the option to utilize all the cores of a machine efficiently. With node, we can use JavaScript even for server side scripting thereby, helps in code reuse, productivity and above all speed. Node is faster than PHP, Ruby or any other server side scripts.

2.4.5. Evaluation Criteria

Hadoop, even though is the king of analyzing big-data, is a batch processing method. It is enough for a lot of operations, but for our primary purpose, we need to do real-time processing on the streaming content received from the data sources. MapReduce operations even though highly scalable can take a lot of time to execute. It is said that Twitter generates around 140 million tweets are generated a day and analyzing this data in real time can create endless possibilities. Since, Twitter is one of our main sources, we need to another real-time data processing system.

Storm and S4 are both real-time data processing systems but S4 is an Apache Incubator project and is still in its early stages of development. Storm is also a viable option as our processing system but it requires much more configuration and since it is also still in its early days, documentations for setup are also insufficient. But Storm will be our second option as the processing system.

So, we will be using node.js as our primary (mandatory) real-time processing system since it is really fast and provides high performance. Another advantage of using Node.js as our engine is that we can use JavaScript to do all the scripting needed for our application, all the way from the server to the client, i.e. with node.js, we can code an entire application with JavaScript including the Server and Client. This in turn helps us to use the same codes and variables throughout the application. Another major attraction of using node.js is the proactive community that is constantly working to develop new and all sorts of modules.

An advantage that node applications have over Storm or S4 is the simplicity of linking a user input or action from a web page to the server application. The server application responds almost immediately to the change.
2.5. Querying Data from NoSQL

To read data from a NoSQL database or Hadoop, programmers can choose from a handful of high-level languages and frameworks that allows data analysis to be performed in an easy way and faster debugging. The hive language from Facebook and Pig from Yahoo! are some examples of query languages used for Hadoop. In addition to this, we can also use Python, JavaScript, R, etc. Tasks written in these languages are compiled down into a series of MapReduce jobs for execution on Hadoop nodes. It has been reported that 90% of Facebook’s jobs are run from these high level languages rather than hand-written map reduce jobs. Some of these high level languages are also similar to SQL.

A number of Hadoop ecosystem projects have been developed to provide more familiar paradigms to taking advantage of Hadoop. Apache pig allows developers to write data analysis programs in a high-level language, called Pig Latin, custom made for exploring big data. Apache Hive presents a data warehouse interface on top of Hadoop, allowing users to issue queries in HiveQL, which is similar to SQL.

2.5.1. Apache Pig

Apache Pig (2012), Pig is a platform for creating data analysis applications for analyzing big data in a high level language along with an infrastructure to evaluating these applications. Pig programs can be modified to enable parallelism, which will enable it to handle big data.

Pig’s infrastructure layer consists of a compiler that produces a sequence of Map Reduce programs, for which large scale parallel implementations already exists. The language layer consists of a language called Pig Latin, with the following key properties:

- **Ease of Programming:** Simple parallel execution of data analysis jobs can be achieved easily. Complex tasks are encoded as data flow sequences that make it easier to understand, write and maintain.
- **Optimization Opportunities:** Automated optimizations are performed when the tasks are executed.
- **Extensibility:** Users can define their own functions to perform special purpose processing.
- **Supports any kinds of data including those with partial or unknown schemas**, unstructured or semi-structured data.

A simple example of Pig in Hadoop for searching big data records which satisfy a certain search criterion (Known as grep in Linux) is as shown:

```
messages = LOAD 'messages';
conf = FILTER messages BY $0 MATCHES '.*CONFIDENTIAL.*';
STORE conf INTO 'private';
```
The first line simply reads the test data set (message log) into a bag, which represents a collection of tuples. Then we filter this data by the entry in the tuple, shown as $0 with a regular expression, looking for character sequence CONFIDENTIAL. Finally we store this bag, which now has all the tuples that contains CONFIDENTIAL into a new file called private in the host file system. From the above example we can clearly see that this small script performs the operation easily, which would require considerably more coding and effort if implemented in MapReduce. This makes it considerably easier to learn Hadoop and begin working with data.

2.5.2. Apache Hive

Apache Hive (2013) states that Hive is data warehousing system for Hadoop that enables ad-hoc queries, easy data summarization and analysis of large datasets in Hadoop compatible file systems. Hive provides a mechanism to project structure onto this data and query the data using a SQL like language called HiveQL. It also allows traditional map/reduce programmers to plug in their custom mappers and reducers when it’s difficult to express the logic in HiveQL. Hive is based on SQL but we can only read existing files in HDFS, it doesn’t come with UPDATE or DELETE support. It focuses on the query part of SQL but has its own additions and changes to use the MapReduce process efficiently. Hive is easier to learn for someone who is familiar with SQL and obtain the results faster. Hive defines the Tables beforehand and stores the schema in a either shared or local database. It comes with a built-in Derby instance to get you started quickly. If the database is local then only you can run Hive specific commands. If it’s shared, then others can also run these or they would have to setup their own local database copy. Types are also defined upfront and supported types are int, bigint, boolean, string and so on. There are also array types that can handle specific fields in raw data as a group. Hive takes a long time to return results for queries even for a very small data set. Therefore, Hive cannot be used for online transactions processing and does not offer real-time queries and row level updates. It is suitable for batch jobs over large data sets of immutable data.

2.5.3. Mongoose for MongoDB

Mongoose is a Node.js asynchronous library that acts as an additional abstraction layer on top of MongoDB installation. It links MongoDB with Node.js. It provides a schema-based, easy-to-use solution for modeling application data and has built-in validation, type casting, query building, logic hooks and more. It provides MongoDB object mapping similar to ORM (Object Relational Mapping) with a familiar interface within Node.js, i.e. Mongoose translates data in the database to JavaScript objects for use in your application.

When using Mongoose, we need to setup a connection first. We can create multiple
connections to different databases using Mongoose. If we are using only one database, we can connect to that using the code:

```javascript
Mongoose.connect
```

To create additional connection, we need to use the code:

```javascript
mongoose.createConnection
```

Both the above commands take a mongodb:// URI or parameters (Host, Database, Port, Options).

```javascript
var mongoose = require('mongoose);
mongoose.connect('mongodb://localhost/my_database');
```

Once the connection is active, the open event is fired on the instance of Connection. If we were using mongoose.connect, the connection is mongoose.connection. Otherwise, mongoose.createConnection return value is a connection.

Next, we need to setup a Schema and model to work with the data that will be stored in our database. Schemas define the structure of documents in a collection in database and models are used to create copies of the same data which will be stored in documents.

```javascript
var Schema = new mongoose.Schema({
title: { type: String }
});
```

The model is created using the Schema as the structure with the following code:

```javascript
var Movie = mongoose.model('Movie', Schema);
```

### 2.5.4 Evaluation Criteria

Since, we will be using MongoDB and Node.js as our primary NoSQL database, we will Mongoose to query the data from MongoDB and check for matches in the Node.js application.

Both Pig and Hive are good in query processing and supports joining, sorting or ordering data dynamically. Hive has a structure similar to SQL so it will be easier for people who are used to SQL. One advantage Pig has over Hive is that Pig has COGROUP that allows OUTER JOIN’s and so. They both have a shell that allows querying specific things or running the actual queries. As the optional requirement, if we setup a Hadoop cluster, we can either use Pig or Hive to query data from Hadoop.
2.6. Knowledge Discovery

According to Begoli E. and Horey J. (2012:215), “A set of activities designed to extract new knowledge from complex datasets is called Knowledge Discovery from Data (KDD).” The knowledge discovery process has the following steps:

- Data Collection, Storage and Organization practices.
- Understanding and effective application of modern data analytic methods.
- Understanding the problem domain and nature, structure and meaning of underlying data.

The Internet is now the most used communication medium for latest news and to collect spilled secrets. The web has become a part of and a reflection of the real world. Following what people are talking about and analyzing what is influencing people, can help to identify trends and latest news and before it is recognized by the world.

Gloor et al (2009: 215) imposes that the internet can be classified into three categories or information spheres:

1. **The Web At Large**: Wisdom on crowds.

2. **Blogosphere**: Represents the Wisdom of Experts. The number of blogs and bloggers are growing exponentially. Bloggers are individual experts who expresses his or her private opinion. Since, experts are not always right, it is risky to just depend on a single opinion but combining the wisdom of many experts can give a collective opinion on a topic.

3. **Forums**: Wisdom of Swarms. They contain the latest and focused information of a certain subject. The huge “swarms” of people represents the collective opinion of those who are interested and care about the topic. People discuss on a topic between each other.

Analytics is a weird business when it comes to read / write characteristics and data access patterns. In CRUD applications and mobile apps and e commerce software, the read/write characteristics are as shown:

![Typical Data Access Pattern](Figure_17_Typical_Data_Access_Pattern_Wikipedia.png)

This figure just explains that data is read far more times than it is written. This is why relational databases and document databases cache frequently read items into memory. But in the case of data analytics, it is the
opposite scenario.

**Analytics Data Access Pattern**

(one read for thousands of writes)

![Analytics Data Access Pattern](Wikipedia)

For Analytics, the database is accessed and writes are made from several different sources and database reads are very less. Data is written in multiple orders of magnitude more frequently than its read.

### 2.7. Sentiment Analysis

Nowadays, businesses and governments are tracking the contents of social networks, tweets and blogs to perform sentiment analysis. Data engineers are studying tweets and social data to understand how information of various forms spreads and how it can be more effectively utilized for good.

Kalwar S. et al (2011:738), Business usage of online social networks is a result of intense development. Member information of a community can be treated as a basis for correct identification of its needs and as a result of adjusting personalized marketing messages. The data in a social network can be divided into:

- **Individual data:** User related data such as age, location, sex, etc. It is also the data that describes the relations between the members (e.g.: number of comments on blog, number of invitations, frequency of contacts with others, etc.

- **Relational Data:** Data describing relations between the members (e.g. frequency of contacts with other users, number of invitations, number of comments on the blog etc.). These data types have behavioral character, as it’s a direct result of real user behavior.

According to Kalwar S. et al (2011: 738), “ User reactions play an essential role in interacting with the Internet. It is understood that feelings are subjective experience of users aroused from different emotions. As feelings changes based on time, circumstance, people and environment, it is extremely difficult to assess feelings objectively.” User’s
interaction on the Internet is based on contents (text, audio and video materials) and context (past experience, surroundings, circumstances, environment, background, or settings). User reaction can be a result of the user’s feelings and thoughts.

Kalwar S. et al (2011: 738) suggests that an emotion of a user on the Internet could be divided into two parts, thoughts and feelings. Both of them might be related. The figure shows a simple categorization of emotions. Nevertheless, what you think is not necessarily, what you feel and what you feel is not necessarily, what you think. Therefore, the feelings and thoughts may be inter-related but not completely related. There are several keywords associated with feelings such as anxious, depressed, angry, sad, nervous, jealous, happy, joyful, cheerful, excited, disgusted, frightened, mad, irritated, panicly, discouraged, uneasy, frustrated, and tense. On the contrary, it is hard to find keywords associated with thoughts. A famous statement from Descartes, "cognito ergo sum" or, "I think, therefore I am". People often use, “I think…, in my opinion, and in my viewpoint” which in simpler terms that might relate to positive, negative and neutral for some events, activities and things.

![Figure 19: Categorization of Emotions (Kalwar S., 2011)](image)

Our seven types of Internet anxiety are based on user’s experiences and feelings. It is understood that users/human feelings are hard to extract. Suppose an anxiety could be broadly categorized and the categorization is based on positive, negative and neutral feelings.

A. **Positive feelings**: First, it is important to categorize the Internet anxiety based on positive feelings. The keywords associated with positive feelings could be cheerful, excited, joyful, and happy and so on.

B. **Negative feelings**: Second, it is also important to categorize the Internet anxiety based on negative feelings. Some of the keywords associated with negative feelings could be anxious, depressed, angry and sad.

C. **Neutral feelings**: Third, it is important to categorize the Internet anxiety based on neutral feelings. Some of the keywords associated with neutral feelings could be “I feel nothing, I do not know, and I cannot say what is my feeling right now”.

As per Ronen Feldman (2013: 83), Sentiment analysis can be of the five different kinds:

1. **Document-Level**: This type of sentiment analysis is performed on the entire
document. It assumes that the whole document will contain only one emotion. Here we will calculate the semantic orientation (SO) of these phrases is above a threshold and is then classified as positive or negative. These phrases can be selected by two approaches: a set or predefined patterns or a lexicon of sentiment words and phrases. To determine the SO, we can calculate the difference between the PMI (Point-wise Mutual Information) of the phrase with two sentiment words. PMI \((p, w)\) measures the statistical dependence between the phrase \(P\) and the word \(W\) based on the co-occurrence in a document or on the web.

2. **Sentence-Level:** In this type of sentiment analysis, we will assume that the same document can contain different opinions. Here, we will assume each sentence of the document has an opinion. Then, we need to find out if a sentence is objective or subjective since the objective sentences are much harder to analyze. Subjective sentences can then be further analyzed to find whether positive or negative.

3. **Aspect-Based:** This analysis takes into account the different aspects or attributes of a document or sentence. For example, a user may comment on the design, build or UI of a certain product. Simply classifying this document as positive or negative will result in the loss of a lot of valuable information. One known method to perform this is to get all the Noun Phrases (NP) from the document and then find the frequency of the NP.

4. **Comparative:** In some cases, the user will just simply compare a product with another. Here, we will have to find out the sentiment by taking both the products and each of their opinions.

5. **Sentiment Lexicon Acquisition:** Sentiment lexicons are the most vital resource for a sentiment analysis algorithm. These lexicon can be obtained in a variety of ways, namely:
   - **Manually:** The lexicons are coded by hand. This is not practical for each domain, as it will require its own lexicons and a big effort.
   - **Dictionary-Based:** A set of seed words is expanded using resources like WordNet. This starts with a small set of seed words for a single domain. This is then expanded using WordNet’s antonyms and synonyms. One of the methods is to find the distance between two terms, which is the length of the shortest path between two terms in WordNet. The SO of the term can then be found out using the proper formula. One disadvantage of dictionary algorithm is the lexicons are domain independent so it does not contain any domain specific lexicons.
   - **Document-Based:** A set of seed words is expanded using large number of documents from same domain. This method enables us to find the sentiment of related words if we know some of the words in a particular sentence. For example, “iPhone sales are growing and fast.”, if we have a positive score for ‘growing’, we can assume ‘fast’ is also positive.

In our project, we will use a list of lexicons to compare each word in the text with each word in the lexicon list. We will use the unigram lexicon list obtained by the lexicon acquisition method of Sentiment analysis in the intriguing research of Saif M. (2013).
2.8. Visualizing the Data

2.8.1. Processing.js
Processing.js is a project designed for the web using the processing visual programming language. It uses web standards without any plugins to make digital art, data visualizations, interactive animation, video games, educational graphs, etc. The code is written in processing language in the web page and the rest is done by processing.js. It started as a Java based open-source programming language that helped the visual design and electronic arts communities learn the computer programming basics in a visual context. Processing.js allows processing code to be run by any HTML5 browser in a <canvas> element. It brings best of visual programming for the web.

2.8.2. Google Charts API
Google Charts API provides a wide range of charts from scatter charts to Tree Maps in an easy to use package. In Google’s own words, “Use the same chart tools Google uses”. The charts are also highly customizable with an extensive range of options to change the look or feel. It also used HTML5 and SVG (Simple Vector Graphics) for rendering the data. These charts are also compatible with iOS and Android devices without the need for any additional plugins. Google Charts API also provides support for dynamic data for querying data from different data sources using a variety of data connectivity tools and protocols.

2.8.3. Google Maps API
The Google Maps API allows us to embed Google Maps on our web pages. It is faster and also works on mobile devices. There are a lot of options available to customize the maps to our specific needs. Google also provides support for multiple overlays on top of the maps. There are also a number of overlays created by other developers for use with Google Maps API.

2.8.4. AJAX Graphs
There are a lot of AJAX graphs available on the web. We can also make use of one such solution to visualize our data. Flot.js is an interesting AJAX library providing dynamic graphs, pie charts, bar charts, etc. Some links for other AJAX graphs are provided in the appendix.

2.8.5. Evaluation Criteria
We can use any of the methods mentioned above to visualize the processed data on the web page. Since, setting up a Processing canvas requires lot of background learning and expertise, we won’t be using it for this project. We will use Google API’s in an interesting way to display the processed data on the web page since it is far more superior to any other AJAX graphs found on the web.
2.9. Methodology

Now, let us briefly discuss the methodology that is used for this project.

We will be using the Agile software development methodology for our development. It is set of methods where the requirements and solutions evolve using iterative and incremental development through self or other functional groups. It supports adaptive planning, time-boxed iterative approach and enables rapid response to modifications. There are many methods of Agile development but most of them allow development, collaboration and process changes throughout the lifecycle of the project.

As per Qureshi M. (2011), “Agility means responding to changes quickly and efficiently. Possible changes required in project are budget, schedule, resources, technology, requirements and team. These are the ‘reacting’ changes on which agile models focus by delivering the first increment in couple of weeks and complete software in couple of months. There are several criteria to classify projects such as size, complexity and mission criticality.”

Agile methods splits jobs into small parts with minimum planning and does not include a long-term planning. Iterations are short time periods (Time-boxes) that last one to four weeks. Each iteration requires planning, requirement analysis, design, coding and testing. This reduces the overall risk and allows the product to make sudden changes.

We can consider our mandatory requirements as iterations, where we will consider each requirement and perform all the above steps for a single requirement. Once, this is properly working, we can move to the next requirement. The advantage of using Agile method is that the software design can change in between the initial design and the module design.
2.10. Additional Research

These are some additional resources, which we may or may not use in our project.

2.10.1. PHP

PHP is a highly popular general-purpose server-side scripting language for Web development. It can create dynamic HTML pages. PHP stands for PHP: HyperText Preprocessor and can be embedded directly into an HTML file. PHP code is interpreted by a web server and generates the resulting web page. An example of a PHP code is as shown:

```php
<?php
Echo "Hello World";
?>
```

The PHP code is expressed inside delimiters as shown. Variables in PHP are case-sensitive and prefixed with a $ symbol and they do not need to be defined in advance. PHP has the standard if conditionals, for and while loops along with many others.

2.10.2. HTML5

HTML 5 is the latest version of HTML that is still under development. It is going to be the new standard. Most of the browsers support the new HTML 5 elements and APIs and adds the latest of HTML5 to their latest version. Some of the new features of HTML 5 are as follows:

1. New Form Elements like <article>, <footer>, <header>, <nav> and <section>.
2. New Form controls like calendar, date, time, email, URL and search.
3. New Attributes.
4. New Input Types.
5. Automatic Validation.
6. Uses CSS3.
7. Inbuilt support for Audio and Video.
8. 2D/3D Graphics.
9. Local Storage.
10. Local SQL Database.

There are new elements like headers, footers, menus, sections and articles. HTML 5 has new tags like <video> and <audio> for embedding video and audio into a webpage. Graphics are supported with the <canvas> element for 2D drawing, Inline SVG and CSS3 2D/3D. HTML 5 doctype is declaration is very simple and is as shown:

```html
<!DOCTYPE html>
```

HTML5 makes web application development simple, by incorporating,

- Local Data Storage.
- Local File Access.
- Local SQL Database.
- Application Cache.
- JavaScript Workers.
2.10.3. CSS3

CSS3 is split into modules. The old specification has been split into smaller pieces and new ones are added. The most important modules are:

- Selectors.
- New Properties.
- Rounded Corners.
- Shadow Effects.
- Downloadable Fonts.
- Box Model.
- Backgrounds and Borders.
- Text Effects.
- 2D/3D Transformations.
- Animations.
- Multiple Column Layouts.
- User Interface.

HTML 5 uses CSS3. CSS3 is backwards compatible completely, so there is no need to change existing designs. Browsers will always support CSS2.

2.10.4. JQuery

JQuery is lightweight JavaScript library. JQuery makes it easier to use JavaScript on webpages. JQuery makes coding easier by wrapping many lines of code into a method that we can invoke in a single line. JQuery makes complicated JavaScript things like DOM manipulations and AJAX calls simple. JQuery has the following features:

- HTML/DOM manipulations.
- CSS manipulation.
- HTML event methods.
- Animations and Effects.
- Utilities.
- AJAX.

2.10.5. Meteor.js

Meteor is an open-source JavaScript platform for creating high quality web apps in a very short time. It is possible to write an entire app in JavaScript. Client and servers have the same API available including database APIs. Pages will be updates as soon as the data in the database changes. Client coding can be done similar to server coding with direct access to database without the need for REST endpoints. Changes made by the users are reflected immediately without any need to wait for the server. Updates can be pushed while users are connected.

2.10.6. Express Framework

Express is flexible and minimal framework for creating Node.js web applications with a proper set of features for building single or multiple page web applications. Express provides all the basic features needed for a web application without spoiling the beauty of Node.js.
2.10.7. **Socket.io**
Socket.IO is a really interesting and important piece of puzzle while developing a node.js application. It is a JavaScript library creates low latency connection between the server and the client by using different transport mechanisms. It aims to makes real-time applications possible in any browser or mobile device. Socket.io chooses the transport mechanism automatically depending on the user’s client. For example, if the browser does not support Web Sockets, it will use long polling to fetch data. We will be using Socket.io a lot in our project!

2.10.8. **Open Weather Maps API**
Open Weather Maps API provides weather information by just providing the geographic coordinates of any location on the Earth. The data can be fetched in JSON, HTML or XML form. The weather information is obtained from weather stations. It provides support for Geocoding to find cities by name, zip code or country. It provides us with different kinds of weather maps such as clouds, rain, temperature, wind, etc. Open Weather maps API does not impose a request limit on queries like other similar services. We will use this API in our application, to obtain the weather information for the region around a tweet or news article. A module for this is already available in Node.js.

2.10.9. **Backbone.js**
Backbone.js is a MVC model for a web application development. It provides structure to web applications by giving:
- **models** with custom events and key-value binding
- **collections** with rich API of numerous functions.
- **views** with event handling and connection to existing API’s on RESTful JSON interface.

Here, we represent all our data as Models that can be created, destroyed, validated and saved to the server. When a UI event causes a model’s attribute to change, the model triggers a “change” event. All the views linked to that model can be notified of the change, so that they update the view accordingly. In a backbone application, we don’t have to write the code to link the DOM to find an element and update its HTML. The view updates by itself, when there is a change in the model.

2.10.10. **Python**
Python is a free to use open-source programming language that lets you work more quickly and integrate your systems more effectively. Python is easy to learn and almost immediate gains in productivity and lower maintenance costs. Python runs on Windows, Linux/Unix, Mac OS X, and has been ported to the Java and .NET virtual machines.

2.10.11. **R**
R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and Macintosh. R is great at plotting graphics, data analysis, fitting statistical models that use data that files it in the memory of the computer.

2.10.12. **Bootstrap**
Twitter Bootstrap is a sleek, intuitive and powerful front-end framework for easier and faster web development. Bootstrap is equipped with HTML, CSS and JS for different kinds of web applications.
2.10.13. AWS
Amazon Web Services (AWS) is Amazon’s cloud offering which includes a wide range of products such as EC2 (Elastic Cloud Compute, Virtual Servers), S3 (Simple Storage Services), DynamoDB (NoSQL), etc. The main attraction of using AWS for our server and storage needs are horizontal (Adding a new Server) and vertical scalability (increase the CPU, Memory, etc.) is just a click away.

Amazon is also providing a Free tier for all its services. This web application will be hosted on AWS and forwarded from my domain. Since we will be using the free tier the application runs a bit slow. It will be accessible at:

http://54.213.89.180:8081/ or http://aws.ronlabz.com:8081/
5. Requirements Analysis

The aim of this project is to perform real-time analytics on social data to extract valuable information from it including the sentiments or the emotions involved. The processed data then need to be displayed on a webpage in an easy to understand and interactive way to the user.

3.1. Objectives

The following are the main objectives of this project:

1. **Data Collection**: The first objective of this project is to collect social data. This can be accomplished through the use of streaming APIs, RSS Feeds, etc.

2. **Data Processing**: The collected data has to be analyzed in real-time by the processing system to extract valuable information from it. We will perform multiple operations on the collected data in this stage.

3. **Data Stores**: Setup the data store needed to handle the storage requirements of the data involved during the processing stage. This can also be extended to store the collected data.

4. **Data Visualization**: The information from the processing system then needs to be displayed in an interesting and simple manner on a web page. The visualization should be able to convey the processed data effectively.
3.2. Mandatory Requirements

The mandatory requirements of this project can be summarized as follows:

1. **Data Extraction:** *In this step, we will have to collect social data.* According to the evaluation criteria mentioned in the literature review, we will use Twitter’s streaming API to obtain the needed data input. Twitter’s streaming API will give us a sample of all the public tweets happening around the world in real-time. *At the end of this stage, this data should be available in the processing system.*

2. **Transform Data:** The collected data can have different formats, for example, the result from Google API will be different from that returned by the Open Weather Map API. Another example is the case of a tweet, where the tweet has to be split up into an array of words and perform additional modifications like making it all lower-case. *Therefore, in this stage, we need to clean, parse and prepare the collected data for our processing system.*

3. **Setting up Data Stores:** In this step, *we need to setup a data store to save all the data needed for the processing system.* Since, the sentiment lexicon list we will be using, contains more than 100,000 words, we will need to setup a NoSQL database to allow for fast lookups with good performance. Therefore, as per the evaluation criteria, *we need to setup MongoDB for our storage needs and load the entire lexicon list into it.* We can also load all the additional data needed for processing into it.

4. **Real-time Processing:** All the data collected and prepared in the previous steps then needs to be fed into a system that can process this data in real-time. This system should then prepare the final data to be displayed on the page. From the evaluation criteria, *we have decided to use Node.js JavaScript engine as our processing system.*

5. **Perform Sentiment Analysis:** The main goal of this project is to find the sentiment or emotion involved in the data. We will execute sentiment analysis using one of the methods described in this report, the sentiment lexicon acquisition method. Here, we will use a known list of words with a sentiment score to compare our data and find the sentiment inside the data. This list will have to be stored in the data store and compared with each word in the tweet. For Example, keywords like happy, joy, etc. can mean that the tweet contains a happy emotion and vice versa. *This analysis will be performed with the help of both the processing system and data store that is already set up.*
6. **Additional Node Modules:** We then need to setup all additional modules needed for node.js to work with all the data and processing requirements of this project. Some of these modules are Async, cluster, MapQuest.

7. **Web Page:** Now that we have the processed data ready, we need to create a web page that can visually represent this data. This page should display dynamic data that will update the contents when new information is available. *We can make use of HTML5, CSS3 and JavaScript to complete this web page.*

8. **Visual Representation:** The data needs to be visually represented in an esthetic and appealing manner to the end user on the web page. It should convey the underlying information clearly to the user. The processed data from the processing system needs to be directly fed to this visualization. *We can make use of Google APIs or any other AJAX graphs discussed in the review to complete this step.*
3.3. Optional Requirements

1. **Additional Processing:** In this step, we can setup an additional data storage system and batch process the collected data to extract more valuable information from it. From the literature review, we can use Hadoop for this purpose using the MapReduce algorithm. For example, we can dig more into why people are sad or angry and may be come up with a solution for the problem.

2. **Alternative Real-time Processing:** Since, the main part of any research is benchmarking all the solution by ourselves, it would be better to try setting up another real-time processing system such as, Twitter Storm, which is our second option in our literature review for real-time processing. The storm implementation can then use the Twitter stream as a spout and we can perform the required processing.

3. **Data Storage:** The data storage mentioned in the mandatory requirement can be extended or changed to incorporate the data collected from the first step. This data can then be used for further processing.

4. **Multiple Sources:** As the mandatory requirement, we will only be obtaining the data from the Twitter API but this can be extended by adding more data sources. Other sources as mentioned in the Literature review are RSS feeds, Crawlers, etc. *RSS feeds can be fetched from news sites and added to our page.* A custom web crawler can be created to go through blogs and social networks such as Facebook to extract real-time data.

5. **Additional Sentiment Analysis:** The sentiment analytics performed can be extended to incorporate more features. For example, instead of simply detecting happy or sad sentiments, we can find out people who are angry. We can also make use of the bigram lexicon list to perform more analytics on the data.

6. **More Visualization:** In addition to providing just one view of the data, we can maybe perform additional processing and *display the new data on another page in an entirely different manner.* We can also make use of Processing.js to create more interesting visualizations.
4. Design

A simple data flow model for this project can be displayed as follows:

Data Source: In this stage, we will setup the data source that will be used to collect the data. We will setup Twitter’s streaming API for this purpose. It provides us access to a variety of public information. We can also try to create a web crawler that will collect the required information from blogs and forums. The data source will be connected to the processing system through adapters or modules.

Data Processing: The data processing will be done using Node.js JavaScript engine. Node.js is an event-driven model that can provide high performance computations for our purpose. The data from the source will be linked to Node using the required modules. We can also try to setup a Storm cluster that will also do real-time processing. Hadoop can also be setup to be used as a secondary batch-processing engine to extract more information from the data using MapReduce algorithms.

Data Visualization: The processed information must be display on a web page using graphs or charts. The web page itself maybe created with HTML, CSS and JavaScript. The visualization part can be fulfilled with Google Maps APIs and Google Charts API. More details on the UI design will be discussed in the UI Design chapter.
5. Professional, Legal, Ethical, and Social Issues

Web mining is the technique of discovering and extracting data from the Internet. While web mining can be of great use for businesses, it also poses a threat to the privacy and individuality of a user. Web mining makes it harder for a user to control the amount of data that he shares on the Internet. Web mining is capable of using this data, combining it with other data and use it in an entirely different context. It can also trace a web user and analyze their actions even without their knowledge.

Another main concern is the personal data of the user being widely publicized on the Internet. We need to respect the privacy and individuality of the user. It should be protected so that people are treated fairly. People should be made aware of the dangers of ethical issues involved with big data and web mining. This invisible gathering of information is a big threat to the privacy of the user as personal data can be misused or is used in an entirely different context. A person does not have the opportunity to approve or deny this data collection or its use.

Another ethical issue when it comes to Big Data is public acceptance. People perceive big data as the Big Brother or the beginning of the machine apocalypse. The terms of collection and information storage must be clearly stated to the user. The user should be aware what data is being collected and how it is going to be used. He should be able to give consent to use this data. Big data has to convince people that it can be trusted and can lead to a new Internet experience.

In our research, we will be dealing mostly with the public data that is available on social networks. The main issue we have to consider is the privacy of the data being collected. We should not collect any private information of a user throughout this project. Another concern can be data storage. The data stored for this research has to be removed after its use in a timely fashion.

We are using Twitter as our main data source. All the tweets that we get through the API are public tweets. Twitter provides a user and option to make a tweet private when they are tweeting. This tells us that the data that we acquire for processing are all public and there are no major issues to be considered (Twitter).
6. Project Plan

The project plan is as displayed in the Gantt chart on next page. I had proposed a timeline of 65 days from the third of May to the first of August. There was also some slack time provided in the case of small delays.

I have initially setup a Linux server for all the testing purposes. Then installed all the softwares needed for our purpose such as Node.js and MongoDB. I also setup the streaming endpoints and connections from the twitter API’s to Node along with other modules that we were needed during development like Mongoose, Async, Cluster, etc. We could also have created a crawler during this phase. After the software setup was complete, I set up the data processing in Node.js. The data processing could be done in two stages one being optional. The data can be processed in real-time by creating and setting up the processing on the Node.js engine, this stage is completed as per the requirements. We could also setup batch data processing on Hadoop to extract more information from the input data, which was not done due to time constraints. This was an optional requirement. Once the data processing was successfully setup, the next thing we had to deal with was setting up Socket.io to enable Server-client and client-server connections. Then we needed to load all the processed data into our webpage using these sockets. We also setup additional processing in the browser as the next step. This additional processing is mostly for the second view and since we are using Node, we can also move this code to the server depending on the loading on the browser. The next step was to setup the first view for displaying the data. Here, we have used the Google Maps API with some additional overlays. We also setup a second view to display the data in an Infographic view so that it includes more concise information in a simplistic interface. The last step is to test and evaluate the overall functioning of the application to make the necessary changes and modifications.

Since this was entirely a software-based project, there are literally no risks involved since we won’t be dealing with any dangerous or harmful devices. So, there is no need for any risk assessment.
Real-time Pictographic Representation of the Semantic Web – Dissertation Report

Figure 21: The Project Plan
7. **UI Design**

Our application has basically two views,

1. **The Map View**: The map view shows the tweets in real-time on a map with the profile picture of the person.
2. **Infographic View**: The Infographic view displays some interesting facts and data in a simple textual manner.

### 7.1. Map View

In this view, we will use Google Maps API to load a world map onto our page with custom styles. The map view should show the tweets at the point of their origin when they happen.

The basic layout of the Map View can be as shown:

![Map View Layout](image)

*Figure 22: the Map View Layout*

When the browser receives a tweet from Node.js, it is added to the Map as a Marker. The marker needs to display some information about the tweet and user. We will use the marker to convey the sentiment of the person.

![Marker Design](image)

*Figure 23: Designing the Marker*

The custom marker appears at the location where the tweets happen. This marker should contain the profile image of the person and should also include the current emotion of the person. We will use a simple design as shown to provide the following info.
When the user clicks on a Marker or tweet on the Map, it should present the user with an Information window that displays some information about the tweet along with some additional data such as the weather at the location or even the device of the user. The basic layout for this info window is as shown below:

![Figure 24: Designing the InfoWindow](image)

### 7.2. Infographic View

The UI design for this view will include the use of Twitter Bootstrap and Google Charts. We will use both pie charts and bar charts to complete this view. This view will also provide a lot of other details and facts in plain text. The values and charts should be dynamically updated depending on the information received from the server. Some of the information that can be included in this view,

- Chart of all sources of tweets.
- Mapping tweets and sentiments to weather.
- Computing Top Hash Tags.
8. Development

In this chapter, we will discuss some of the technologies used during the development of this project along with the important code snippets. Let us begin with the tools used for development,

8.1. Tools Used

**Node.js:** An introduction to node.js is provided in the literature review. We will use node.js as our processing engine to do all the processing on the server. It will also function as our HTTP server to server all the page requests. Node.js will be continuously running on the server.

**Express.js:** Express is a framework for node.js. We will use it to deploy our web application. Express does all the work for setting up routes, request handling, etc. Express makes using Node.js easy.

**Connect.js:** Connect is a middleware for node.js engine.

**Cluster.js:** Node.js is designed to work on a single core of CPU. It allows all the processes to share the same server ports. We will make use of cluster to extend node to all the cores of the CPU.

**Async.js:** Async is a utility module for node.js, which is both powerful and effective. Async helps us to work efficiently with the callback structure of the node.js in a simple and organized manner. We will make use of the asynchronous workflow function like Series, parallel, waterfall, each, etc. of the Async library for our application.

**Socket.io:** Socket.io allows us to create consistent connection between the client and server. We will use socket.io to transfer data between the server and client in both ways.

**Mapquest.js:** This is an asynchronous MapQuest API library for Node.js. We will use the geocoding function provided by MapQuest open API to look up coordinates.

**Geocoder.js:** Geocoder is another module for node.js, which provides geocoding using Google API’s. The problem with Google API’s is that it imposes a limit on the requests. So, we will use Geocoder as a backup.

**Geonoder.js:** Geonoder is another module that performs geocoding but support multiple providers. We will also use it in our application.

**Twit.js:** Twit is an asynchronous module for node.js, that links twitter streams into Node.js. It supports both the REST and the streaming endpoints. We will use this module to setup the streaming endpoints and load the tweets into our application.

**RSS Parser:** The RSS parser is an asynchronous library for node.js, which will fetch RSS feeds from sites and parses it into objects that we can use in our application.

**Country-Reverse-Geocoding:** This is a module for node.js that performs reverse geocoding i.e. convert coordinates to countries. We will use it in our application to perform additional analytics on the tweets.

**OpenWeatherMap.js:** Open Weather Maps API module for node.js. We will use this to load the weather information at the coordinates of a particular tweet.

**Jade Templating Engine:** Jade template provides a simple way of creating and writing HTML pages. Jade uses indentations to specify hierarchy of code and do not need angle brackets. Express framework uses Jade by default for rendering views.

**Google Maps API:** We will use this to display the world map on our main view and to create overlays and markers.

**RSS Mix:** This is actually a service offered by site rssmix.com that allows us to mix merge multiple RSS feeds together into one. We will use it to aggregate all the news feeds.
**Day Night Overlay:** This is an overlay for Google maps that provides an overlay of Day/Night over the maps. We will add it to our maps to perform additional analytics. A sample of the result returned by the different modules is given in the appendix.

**Rich Markers Overlay:** Rich markers provide another overlay over Google maps that add support for custom markers. We will use Rich Markers over the default markers in the API since it allows us to customize the marker for our needs.

**InfoWindow:** This is an overlay provided by Google to display information about a marker on the map. We will use it to display more information about a tweet depending on user action.

**Google Charts API:** Google charts provide different kinds of charts for our use. We will make use of the bar charts and pie charts to display different information on our page.

**objSort.js:** This is a handy JavaScript library that performs sorting of an array of objects.

### 8.2. Objects and Data Models

Since our application deals with Twitter feeds, we will initially create a custom class with only the fields we need from the Twitter feed. The custom class, *ctweets*, is defined as shown:

```javascript
function ctweets(ttext, tscore, pflag, coordinates, country, language, source, image, swear_index, weather, rt){
    this.ttext=(typeof ttext === "undefined") ? "defaultValue" : ttext;
    this.score=(typeof tscore === "undefined") ? "0" : tscore;
    //this.positive = (typeof positive === "undefined") ? "0" : positive;
    //this.negative = (typeof negative === "undefined") ? "0" : negative;
    this.pflag = (typeof pflag === "undefined") ? "false" : pflag;
    this.coordinates = (typeof coordinates === "undefined") ? "{}" : coordinates;
    this.country = (typeof country === "undefined") ? "defaultValue" : country;
    this.language = (typeof language === "undefined") ? "defaultValue" : language;
    this.source = (typeof source === "undefined") ? "defaultValue" : source;
    this.image = (typeof image === "undefined") ? "defaultValue" : image;
    this.swear_index = (typeof swear_index === "undefined") ? "defaultValue" : swear_index;
    this.weather = (typeof weather === "undefined") ? "defaultValue" : weather;
    this.rt = (typeof rt ==="undefined") ? "false" :rt;
}
```

This class contains the following properties:

- **ttext:** String. This is the text of the tweet.
- **tscore:** Number. The sentiment score of the tweet.
- **pflag:** Boolean. Indicates whether the tweet is positive or negative.
- **coordinates:** Array of Values. Provides the coordinates of the tweet.
- **country:** String. The country of origin of the tweet.
- **language:** String. The language of the tweet.
- **source:** String. The source or device of the tweet.
- **image:** String. The profile image URL of the user.
- **swear_index:** Boolean. Indicates whether the tweet contains a swear word or not.
- **weather:** Object. Object returned by the OpenWeatherMap module.
- **rt:** Boolean. Indicates whether the tweet is a Retweet or not.

The unigram sentiment lexicon list obtained from the research of Saif Mohammed has four columns, namely:

**Term:** The sentiment lexicon.
**Sentiment Score:** The sentiment score of the term.

**Positive PMI:** The positive PMI value of the term.

**Negative PMI:** The negative PMI value of the term.

Since, this list contains about 100,000 words it is neither practical nor possible to load all of it into the browser’s memory. So, we will make use of our NoSQL database, MongoDB to store this list and perform keyword comparison of tweets and RSS feeds.

We will also make use of a list of swear words to perform additional analysis such as detect people who are angry, who use swear words, etc.

### 8.3. Server Application

We will extensively use the Async module for node in our app.js for effective utilization of CPU and to provision for parallel processing. This module helps to run multiple tasks in parallel or in series and executes the callbacks when all the operations are completed. Because of Node’s asynchronous nature, the Async module is a much needed utility to perform even the most basic tasks in node. It is also one of the most popular modules for Node in the community. We will mainly use four functions of Async,

1. **Async.each:** Iterates through a loop and passed each of the values to a function and executes a callback once its done looping through all the items.
2. **Async.parallel:** Executes multiple tasks in parallel and calls the callback once all the tasks have completed.
3. **Async.series:** Executes the tasks one after the other and calls the callback once all the function have been successfully completed.
4. **Async.waterfall:** Executes the tasks or function one after the other like a waterfall model and calls the callback once everything is completed.

I have used a mixture of all these Async function make efficient utilization of the event loop of node and make the application truly parallel.

We will use Node’s cluster module to make complete use of all the cores of the CPU. On the main CPU, we only fork child processing depending on the number of cores. On the worker nodes, we will perform all the processing works. We are using the parser module for obtaining the RSS feeds and parsing it into our system.

We will then setup Twitter on the worker nodes by providing the consumer_key, access_token, access_token_secret and the consumer_secret (Obtained from the Twitter Developers Page) as follows:

```javascript
var T = new twit({
    consumer_key: 'Fj4PyQOC1yEx0gOiANg',
    consumer_secret: 'XLVDdCiU6UOFWlgAg32I5Kc',
    access_token: 'ri6OGjY8ldZSaiL9c3ltvjmFL2Ju',
    access_token_secret: '11tKeRBq3JgLWcIbzdtq01kUAcavg6Yg'
});
```

Now that we have Twitter ready for streaming, we need to setup Mongoose to connect to our MongoDB instance as follows:

```javascript
var db = mongoose.createConnection('mongodb://localhost/twitterStream/');
db.on('error', console.error);
var tlist =[];
```
We have to initialize a word schema for the data from MongoDB now as shown:

```javascript
var wordSchema = new mongoose.Schema(
    { 
        term: String, 
        sentscore: Number, 
        positive: Number, 
        negative: Number
    });

var unigrams = db.model('unigrams', wordSchema);
```

We will also load a list of swear words obtained from in a similar fashion to look up the angry tweets.

Here, we will be setting up the Express server as shown:

```javascript
//Setup Express
var server = express.createServer();
server.configure(function(){
    server.set('views', __dirname + '/views');
    server.set('view options', { layout: false });
    server.use(express.cookieParser());
    server.use(express.session({ secret: "shhhhhhhhhhhhh!"}));
    server.use(express.static(__dirname + '/static'));
    server.use(server.router);
});
```

In this step, we will setup the errors (404 and 500):

```javascript
server.error(function(err, req, res, next){
    if (err instanceof NotFound) {
        res.render('404.jade', { locals: {
            title : '404 - Not Found'
        ,description: "
        ,author: "
        ,analytics:siteid: 'XXXXXXX'
        ,status: 404 });
    } else {
        res.render('500.jade', { locals: {
            title : 'The Server Encountered an Error'
        ,description: "
        ,author: "
        ,analytics:siteid: 'XXXXXXX'
        ,error: err
        ,status: 500 });
    }});
```

This will render the corresponding views, when the error occurs.
Setting up the routes for our Server requests from clients. We will setup mainly two routes, one for each view, Maps View and the Infographics View. The route for Maps view is setup as shown:

```javascript
server.get('/', function(req, res){
    res.render('index.jade', {
        locals : {
            title : 'Streaming Twitter',
            description: 'Pictographic representation of the Semantic Web.',
            author: 'Ronny Mathew
        }
    });
});

server.get('/info', function(req, res){
    res.render('info.jade', {
        locals : {
            title : 'Infographics',
            description: 'Pictographic representation of the Semantic Web.',
            author: 'Ronny Mathew',
            analyticsSiteId: 'XXXXXXX'
        }
    });
});
```

Additional routes provided for the 404 and 500 error codes:

```javascript
server.get('/404', function(req, res){
    throw new Error('This is a 404 Error');
});
server.get('/500', function(req, res){
    throw new Error('This is a 500 Error');
});
```

The function NotFound() is as shown below:

```javascript
function NotFound(msg){
    this.name = 'NotFound';
    Error.call(this, msg);
    Error.captureStackTrace(this, arguments.callee);
}
```

Now that we have all the necessary steps completed for the server, we can setup the server to listen to a particular port, with the following code:

```javascript
server.listen(port);
```
Setting up socket.io for communication between the server and the client is as shown:

```
var io = io.listen(server);
io.set('log level', 1); // reduce logging
io.set('force new connection', true);
```

Then we will open the Twitter stream and check for English tweets, with the following:

```
var stream = T.stream('statuses/sample');
var lang = "";

stream.on('tweet', function (tweet) {
    //console.log(tweet) //print raw data    lang=tweet.lang;    if (lang=="en"){ ... ] });
```

To perform a word by word comparison with the lexicons in the database and obtain its sentiment score. The code is as shown:

```
unigrams.findOne({ term: word.toString().toLowerCase() }, function(err, result) {
    if (err) return console.error(err);
    res = result;
    callback(result);
});
```

In another parallel task, we will also compute the coordinates of the tweet. This part of code is as shown:

```
function(callback){
    if ((tweet.coordinates!==null)&&(tweet.coordinates!="")) {
        coords[0]=tweet.coordinates.coordinates[1];
        coords[1]=tweet.coordinates.coordinates[0];
        callback();
    } else if ((tweet.place!==null)&&(tweet.place!="")) {
        geonoder.toCoordinates(tweet.place.name.toString(),
        geonoder.providers.google, function(lat, long) {
            if (lat!=null) {
                coords=[lat,long];
                callback();
            } else {
                mapquest.geocode(tweet.place.name.toString(), function(err, location) {
                    if (location) {
                        coords=[location.latLng.lat, location.latLng.lng];
                        callback();
                    }
                });
            }
        });
    }
}
```

Here we make use of the Geonoder and MapQuest modules to obtain the coordinates from the Place value inside the entities object, from time zone or from the user’s location of a tweet, if the tweet does not have Geo enabled. If the tweet has geo enabled, the tweet will contain the coordinates inside itself.
Once we get the coordinates of a tweet, we will pass it to the function, which will use the coordinates to obtain the current weather information at that location using the OpenWeatherMap module.

```javascript
try{
    weather.defaults={units:'metric', lang:'en', mode:'json'};
    if(coords.length){
        weather.now({lon: coords[1], lat: coords[0]}, function(data){
            climate = data.weather;
            callback();
        });
    }
    else{
        climate=null;
        callback();
    }
}catch(err){ console.log(err.message);
    weather.defaults={units:'metric', lang:'en', mode:'json'};
    weather.now({lon: coords[1], lat: coords[0]}, function(data){
        climate = data.weather;
        callback();
    });
}
```

We will also pass these coordinates to the reverse country geocoding function to obtain the country of the tweet as shown:

```javascript
function(callback){
    if(coords.length){
        country = crg.get_country(coords[0],coords[1]).name;
        callback();
    }
    else{
        country=null;
        callback();
    }
}
```

The next function gets the source of the tweet and passes it to an array, which will later be passed to the browser. The different sources, we consider in this project as BlackBerry, Android, iPhone, Web, Mobile Web, iPad, Windows Phone. It then searches for the name of the device in the source as shown:
The next function will compare each and every word of the tweet with a list of swear words and sets the swear_index flag to true if a swear word is found in the tweet.

```javascript
function(callback){
    for(var s=0;s<swearing.length;s++){
        var sword=swearing[s].word;
        if(tweet.text.toLowerCase().indexOf(sword.toLowerCase())!=-1){
            sflag=true;
        }
    }
    if(sflag){
        callback();
    }
}
```
This function checks if the tweet is a Retweet and sets a flag as shown:

```javascript
function(callback){
  if(tweet.text.toString().search(/RT/i)!==-1){
    rflag = true;
    callback();
  } else if (tweet.retweeted){
    rflag = true;
    callback();
  } else
    callback();
}
```

Since, most people does not use the retweet button and will just add the RT keyword, we have also included a check for it too in our code as shown above.

When all the above tasks are completed, we will call the callback method, which is to write all these computed values to an object of our custom class (ctweet). This object is then send to the client or browser through the socket. The code is as shown:

```javascript
function(){                  //Callback Method for 2nd Level Async Parallel
  if(score>0)
    pflag=true;
  else
    pflag=false;
  var this_tweet = new ctweets (tweet.text, score, pflag, cords, country, tweet.lang, src, tweet.user.profile_image_url, sflag, climate, rflag);
  io.sockets.emit ('twitter', this_tweet);
}
```

The next function computes a list of Hash tags from the hashtags array in entities object of a tweet and adds the hash tags to our list. This list contains the hash tagged terms and a weight for each of these term. We will only store 100 words in this list and this function will automatically pop a term if the length of the list is more than 100 words and if its weight is less than a certain value.
var tag;
for (var i=0; i<tweet.entities['hashtags'].length;i++) {
    var f=0;
    tag=tweet.entities['hashtags'][i].text;
    for (var h=0; h<httags.length;h++) {
        var hterm = httags[h].text.toString();
        if (htterm.toLowerCase()==tag.toString().toLowerCase()) {
            f=1;
            httags[h].weight++;
            break;
        }
    }
    if (f==0) {
        httags.push({text: tag.toString(), weight: 1});
    }
}
var hweight=0;
if(httags.length>100){
    for (var w=0; w<httags.length;w++) {
        if(httags[w].weight>hweight){
            hweight=httags[w].weight;
        }
    }
    for (var i=0;i<httags.length;i++) {
        if(httags[i].weight<hweight-3) {
            httags.splice(i,1);
        }
    }
}
w = tweet.weather;
io.sockets.emit('ht',httags);
callback();}]

This function performs a live search on the tweets, it will search for a keyword in the tweet and if present, will send the tweet to the browser instantly. The keyword for search can be set from the browser through the search bar. When a user enters the term inside the text box, this value is transferred to the server immediately.

```javascript
function(callback) { //Search for Keyword
    if (searchword!=""&&searchword!==null) {
        if(tweet.text.toString().toLowerCase().indexOf(searchword)!=-1) {
            io.sockets.emit('sresults',tweet.text);
callback();
        } else {
callback();
    } else {
callback();
}
}, function(ht){ //1st Level CB
})
```
This function uses the user’s position returned from the user agent by using the Geolocation feature of HTML5 to find the country of the user. We will use the reverse country geocoding module so that we can find tweets happening in the same country as the user.

```javascript
socket.on('pos', function(data){
    console.log(data);
    var c = crg.get_country(data[0],data[1]).name;
    console.log(c);
    socket.emit('userc',c);
});
```

### 8.4. Maps View

The maps view displays a map on the screen with markers to represent each tweet. The maps view includes a script.js file that will open a socket with the server and starts to receive the processed tweets from the server. These tweets are then dropped to their corresponding locations on the map using the Google API’s, Rich Marker and InfoWindow overlay. We have also used HTML5 Geolocation to point out the user’s location on the map. The function for Geolocation is as shown:

```javascript
if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(function (location){
        if(location)
            urloc=[location.coords.latitude,
                  location.coords.longitude];
        socket.emit('pos',urloc);
    });
}
```

The Google Maps API is initialized with the following function:

```javascript
function initialize() {
    var mapOptions = {
        zoom: 2,
        mapTypeId: google.maps.MapTypeId.ROADMAP,
        center: center,
        styles: mapStyle,
        scrollwheel: false,
        navigationControl: false,
        //mapTypeControl: false,
        scaleControl: false,
        draggable: false,
        zoomControl: false,
        scaleControl: false,
        panControl: false,
        disableDoubleClickZoom: true
    };

    map = new google.maps.Map(document.getElementById('mapcanvas'), mapOptions);
}
```
Now, we need to initialize the Day Night Overlay on the Google Maps:

```javascript
new DayNightOverlay({
  map: map
});
```

We store the location of the user in a local variable and convert it into coordinates using Google Maps LatLng function as shown:

```javascript
urlLoc = new google.maps.LatLng(location.coords.latitude, location.coords.longitude);
```

When the browser receives a tweet, we need to create the marker along with an InfoWindow on the map. The InfoWindow contains the user’s tweet, profile image and the current weather at their location. We will also assign different border color to each tweet depending on the emotion of the tweet i.e. Green for happy, red for angry and blue for sad tweets.

```javascript
function addOneMarker(tweet) {
  var markerImage = new google.maps.MarkerImage(imageUrl, null, null, null, new google.maps.Size(24, 32));
  if (image != '') {
    markerImage = new google.maps.MarkerImage(image, new google.maps.Size(24, 32));
  }
  if (tweet.pflag) {
    color = '#00FF22';
  } else {
    if (tweet.swear_index) {
      color = '#FF2200';
    } else {
      color = '#0033FF';
    }
  }
  var marker = new RichMarker({
    position: newpoint,
    map: map,
    draggable: false,
    anchor: RichMarkerPosition.MIDDLE,
    animation: google.maps.Animation.BOUNCE,
    content: '<div class="my-marker"
                onclick="alert(tweet.weather[0].main);return false;"
                style="border-radius: 5px; opacity: 0.7; height: 30px; width: 25px; border: 2px solid ' + color + '; background-color: #FFF; background-image: url(' + imageUrl + ');"></div>
                <div type="hidden">+tweet.weather[0].main+</div></div>',
    marker.info = new google.maps.InfoWindow({
      content: '<b>Speed:</b> ' + tweet.ttext
    });
    google.maps.event.addListener(marker, 'click', function() {
      marker.info.open(map, marker);
      markers.push(marker);
    });
  });
}
```

We assign the color on the border based on the simple logic that if the positive flag is set, we can conclude that the tweet is Happy. If it is unset, we will further check if the swear_index is set, if so, we can infer that the tweet is Angry or else Sad.
8.5. Info View

The Infographic view uses Twitter Bootstrap as base CSS. It displays a lot of information on a single page. We will also add Pie Charts and Bar Charts that work on dynamic data in this view. The Infographic view basically contains multiple counters which keeps a count of the number of tweets, number of positive tweets, negative tweets, and so on.

The important piece of code in this view is that one which loads the Google Charts into our page. The code for rendering the pie chart for the source list is as shown:

```javascript
function drawChart() {
  var data = new google.visualization.DataTable();
  // Create the data table.
  data.addColumn('string', 'Source');
  data.addColumn('number', 'Number');
  data.addRows([
    ['iPhone', slist[0]],
    ['iPad', slist[1]],
    ['Android', slist[2]],
    ['Windows Phone', slist[3]],
    ['BlackBerry', slist[4]],
    ['Mobile Web', slist[5]],
    ['Web', slist[6]],
    ['Misc', slist[7]]
  ]);
  // Set chart options
  var options = {'title':'Device List',
                  'width':250,
                  'height':200,
                  is3D: true,
                  backgroundColor:'transparent',
                  opacity:0.3,
                  color:'white',
                  chartArea: {'width': '100%', 'height': '80%'}
  };
  var c = new google.visualization.BarChart(document.getElementById(divid));
  c.draw(data, options);
}
```

This code will create a Google DataTable with two columns, Source and Number. We will then add the rows to this table from another list, which keeps track of the sources.

The code for rendering a bar chart with the countries, positive and negative tweets for a particular weather as shown,

```javascript
function drawWeatherChart(l, divid, text) {
  d =new google.visualization.DataTable();
  // Create the data table.
  var numRows = l.length;
  d.addColumn('string', 'Country');
  d.addColumn('number', 'Positive Tweets');
  d.addColumn('number', 'Negative Tweets');
  var maxvalue = 0;
  for(var i=0; i<l.length;i++){
    maxvalue = l[i].ptw + l[i].ntw;
  }
  for(var i=0; i<l.length;i++){
    d.addRow([l[i].text, l[i].ptw, l[i].ntw]);
  }
  var op = {
    title: 'Tweets when its '+text,
    vAxis: {title: 'Countries', titleTextStyle: {color: 'blue'}}
  };
  var c = new google.visualization.BarChart(document.getElementById(divid));
  c.draw(d, op);
}
```
We are also using a JavaScript library objSort.js to sort an array of objects according to their value. An example of this is where we compute the top 5 languages used.

```javascript
//Language list
var f=0;
for (var l=0; l<list.length;l++){
    //console.log(httags[h]);
    var lang = list[l].text;
    if(tweet.language==lang) {
        f=1;
        //console.log(f);
        //$('#receiver').append('<li>' + w + '</li>');
        //if(wlist[w].weight<=30)
        if(tweet.score>0)
            list[l].ptweets++;
        else
            list[l].ntweets++;
        break;
    }
}
if (f==0) {
    var pt= 0, nt=0;
    if(tweet.score>0)
        pt++;
    else
        nt++;
    list.push({text: tweet.language, ptweets:pt, ntweets:nt});
}
list.objSort("ptweets","ntweets","text");
for(var l=0;l<5;++l){
    $('#langlist').append(list[l]);
}
```

The Infographic view also lists the RSS feeds processed by the server in the order of their sentiment score. We can also make use of the services offered by RSS mix to combine multiple RSS feeds into a single one.
9. **User Guide**

9.1. **The Node Application**

The node.js application uses the Express framework. From the user’s perspective there is nothing much to do at the server side except to start the application by starting it from a console or terminal directly by executing the following code:

```
# node app.js
```

Where app.js is our JavaScript file.

9.2. **The Map View**

This view uses Google Maps API with Socket.io to display live tweets on the Map as they are happening at their corresponding location. The Map View is as shown:

![Map View](image)

A new marker is created for each tweet that is happening. The marker contains the user’s twitter profile image. Changing the border color of the marker as per the following effectively conveys the emotion of the tweet:

- All the Happy tweets are represented using Green color.
- All the sad tweets are represented using Blue color.
- All the angry tweets are represented using Red color.
When the user clicks on a marker, an Information window will appear. This window contains information about the tweet such as the original text, image, source, etc. A screenshot of the Information window is as shown.

There is also an additional overlay provided on top of the Google Maps that displays a DayNight overlay that indicates the areas where it is currently day or night. Analysts can make use of this information to perform further analytics on the data. For example, find out the number of twitter users at night/day, emotion of users at night/day, etc.

A search bar is provided on top of the page to allow live searching of tweets. When the user enters a keyword in the search bar and clicks enter, searching process starts. All the live tweets containing this particular term will be displayed on screen. The latest 10 results will be displayed the search.

9.3. The Infographic View

This view provides additional information derived from the input data in a textual and graphical form. It displays pie chart and bar charts to convey the information effectively. This view is divided into multiple sections, each section is dedicated for a particular information. The first section or the introduction is as shown:

This section displays the number of Happy, angry and sad tweets happening near you. This makes use of Geolocation of HTML5. User needs to enable Geolocation for this to work.
In the next section, we will find out the number of tweets with swear words and how many of them have a positive emotion and a negative emotion:

Now, we can also analyse the swear words people use..

Total number of Tweets with swear words
10
Out of which Positive Tweets
3
Negative tweets
7

Figure 29: Swear Word Analysis

Next section maps the weather with positive and negative tweets. It also provides four different bar charts that display the number of positive or negative tweets from different country according to a particular weather.

Figure 30: Mapping Weather to Sentiments

Here we will analyze the source of the tweet. A pie chart displays the total percentage of each source. This section is designed as shown:

Figure 31: Source Analytics
A list of the most generated Hash Tags and the top 5 languages are also shown,

![Top 10 HashTags and Top 5 Languages](image)

This section lists RSS feeds in the order of their sentiment score:

![Listing the RSS Feeds](image)

This view is currently under development and will soon include lot more interesting details and analytics.
10. Evaluation & Testing

In this section, we will start the evaluation and testing of our application in accordance with the Requirements and evaluation criteria previously mentioned. All the main objectives of this project are completed. A summary of the objectives is as follows:

1. **Data Collection**: Streaming data from twitter and RSS feeds are loaded into the system in real-time and processed.
2. **Data Processing**: The Node.js engine performs all the needed operations on the procured data.
3. **Data Stores**: We have successfully setup and linked an instance of MongoDB with our system.
4. **Data Visualization**: The processed data are visualized on the web page in two views.
10.1. Mandatory Requirements Checklist

Here we will evaluate the mandatory requirement with the evaluation criteria and the final result.

<table>
<thead>
<tr>
<th>Mandatory Requirement</th>
<th>Steps Taken</th>
<th>Technologies Used</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Extraction: Collect Social Data</td>
<td>We have successfully brought the Twitter data into Node.js using the Twit module for node.</td>
<td>Twitter API, Javascript, RegEx, MongoDB, Node.js, Socket.io</td>
<td>Done</td>
</tr>
<tr>
<td>Transform Data</td>
<td>The data is prepared as per the requirement before processing it.</td>
<td>Javascript, RegEx</td>
<td>Done</td>
</tr>
<tr>
<td>Set up a Data Store</td>
<td>We have successfully setup MongoDB and loaded the list of lexicons into it.</td>
<td>Node.js, Socket.io, MongoDB</td>
<td>Done</td>
</tr>
<tr>
<td>Real-time Processing</td>
<td>Node.js has been setup to successfully process the input data.</td>
<td>Node.js, Socket.io, MongoDB</td>
<td>Done</td>
</tr>
<tr>
<td>Sentiment Analysis</td>
<td>We have used the lexicon list with MongoDB and Node.js to successfully complete this task.</td>
<td>Lexicon list from NRC, Javascript, Node.js, MongoDB</td>
<td>Done</td>
</tr>
<tr>
<td>Additional Modules</td>
<td>We have successfully setup all the necessary modules needed for our application.</td>
<td>Async, Cluster, MapQuest, OpenWeatherMap, Express, socket.io</td>
<td>Done</td>
</tr>
<tr>
<td>Web Page</td>
<td>Since we are using Express framework for Node, we have used the Jade templating engine for creating pages/views.</td>
<td>HTML5, CSS3, Javascript, jQuery, Jade</td>
<td>Done</td>
</tr>
<tr>
<td>Visual Representation</td>
<td>We have created two views to display the data, one shows the tweets happening around the world on a Map using Google Maps and the other provides an infographic view of the data.</td>
<td>Google Maps API, Google Charts API, Javascript, Node.js, MapQuest, Express</td>
<td>Done</td>
</tr>
</tbody>
</table>

Figure 34: Mandatory Requirement Checklist
## 10.2. Optional Requirements Checklist

Here we will evaluate the optional requirement with the evaluation criteria and the final result.

<table>
<thead>
<tr>
<th>Optional Requirement</th>
<th>Evaluation Criteria</th>
<th>Status</th>
<th>Technologies Used</th>
<th>Steps Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additional Processing</td>
<td>Setup a Hadoop cluster to perform additional processing on the data using MapReduce Algorithms</td>
<td>Not Done</td>
<td>-</td>
<td>Insufficient time to complete this task.</td>
</tr>
<tr>
<td>Alternative Real-time Processing</td>
<td>Setup a Storm cluster to do real-time processing on the data</td>
<td>Not Done</td>
<td>Storm</td>
<td>I have created a Storm node on a local machine and created a Java application to collect tweets from Twitter API for this purpose. But again, not completed because of insufficient time.</td>
</tr>
<tr>
<td>Multiple Sources</td>
<td>Use data from different sources such as RSS feeds, crawlers, Facebook APIs, etc.</td>
<td>Partial</td>
<td>RSS Feeds, RSS Mix</td>
<td>I have linked RSS feeds into our application. But was not able to connect with crawlers or Facebook.</td>
</tr>
<tr>
<td>Additional Sentiment Analysis</td>
<td>Perform additional sentiment analysis on the data by finding out more information about the tweet</td>
<td>Done</td>
<td>-</td>
<td>We have used a list of swear words from noswearing to find out the tweets that contains swear words and thereby, find out the tweets which contains an angry emotion.</td>
</tr>
<tr>
<td>More Visualization</td>
<td>Provide more additional visualization to obtain more information from the processed data</td>
<td>Done</td>
<td>Google Charts API, CSS3</td>
<td>We have provided a simple infographic view along with the Google charts to provide a second view that displays all the data collected by our system from an analytic viewpoint.</td>
</tr>
</tbody>
</table>

*Figure 35: Optional Requirement Checklist*
10.3. Testing

While testing the application, the following bugs were discovered:

1. The asynchronous callback model of Node.js created a lot of issues initially, even for simple tasks such as assigning values from databases to a variable in our application. This led us to using the Async module in our application. The Async module was further extended to provide parallel processing capabilities on the server.

2. Jade template gives syntax errors at times while modifying HTML. This is caused due to the indentation. When we are using Jade, we need to be extra careful about the indentation and spacing.

3. Socket.io gives a warning message, “client not handshaken client should reconnect” at times and takes more time to setup a connection during start up. Sometimes, the connection is also dropped while connected. Troubleshooting needs to be done with socket.io code.

4. When multiple clients are connected to the server, the streaming speed is reduced. This can be because of the additional loading on the testing server that can be rectified on the production server.

5. Socket.io connection opened by a browser is not closed properly when all the clients are disconnected. The code for opening sockets on the server was checked and found out that the disconnect code was missing.

6. When using the Infographic view, the browser can load up and crash. This can be because all of the processing being done behind the scenes. So, the testing has been done one a module-by-module basis instead of running everything together. We have to move all this processing data back to the server to rectify this.

7. Some tweets or markers appear on the top left corner of the page. These are the markers which does not have any coordinates linked with it. This is because of the API request rate imposed by the geocoding API provider and it returns null values. We are using the geocoding function to find the coordinates of the tweets that do not have the coordinates linked with it.

8. The Weather API also returns null values at times causing the Worker node to crash. An additional step to restart the worker if stopped is provided to enable fault tolerance.

10.4. Additional Work

During the initial phases of development, I was confused on whether to use Storm or Node for processing data. So, for this purpose I have completed the initial set up of a Storm cluster on a Linux system. I have also completed the setup Zookeeper to manage the tasks across different storm servers. Setting up a Storm server requires a lot of prerequisites and I have successfully completed all of it. But while thinking of transferring data between the client and server, I realized node.js is a better solution.

I have also created a Java application to fetch the twitter feeds using Twitter4j library. The code for the same is provided in the appendix.

The next option was to link Storm with node.js so horizontal scaling will be possible because of the lack of time I couldn’t complete those codes. The code to link Storm with node.js is provided in the Appendix.

I have also tried my luck with processing.js for visualizing the data. The processing code that I developed is also given in the Appendix.
11. Conclusion & Future Work

The aim of this project was to perform data analytics on real-time data using the suitable systems and to develop a web application that can display the current emotional state of the Internet. This research has also helped me gain a lot of insights into the underlying concepts of Hadoop, HDFS, MapReduce, Storm and various other technologies. Even though, I have not used some of these technologies, all of this information has also indirectly helped me to come up with the logic and expertise needed to complete this project. This is the first time I’m using Node.js and socket.io and I’m simply amazed and spellbound by the speed and performance it offers. I have also did some research on the latest in web technologies like HTML5, CSS3, PHP and jQuery for developing the web page. I want to thank my supervisor for all the help and support he provided me during this project. I also wanted to thank all my parents and friends who helped me at various stages of development of this project.

Some future work on this project will include the following modifications:

- **Multiple Sources:** Create a crawler to obtain data from blogs, forums, social networks. Also incorporate Facebook Graph API in some way.

- **Hadoop:** Setup a Hadoop cluster for batch processing of the input data and perform deeper analytics.

- **Real-time Information System:** The RSS feeds in the application can be linked with the twitter feeds in a way so that we can detect problems or catastrophes as soon as it happens.

- **Facebook API:** Link the existing Twitter API to Facebook’s Graph API, so that when a tweet is received, we can use that username to search for the same user in Facebook and obtain his public statuses, locations, etc.

- **CSS Customization:** Optimize the CSS templates used for both the views.
12. References


Cattell R. (2010), ‘Scalable SQL and NoSQL Data Stores’.

Facebook, Facebook Developer Tools, [online], Available: http://developers.Facebook.com [15 March 2013]


Humbetov S. (2012), ‘Data-Intensive Computing with Map-Reduce and Hadoop’, Department of Computer Engineering, Qafqaz University, Baku, Azerbaijan


13. Bibliography


Cattell R. (2010), ‘Scalable SQL and NoSQL Data Stores’.

EMC Academics, ‘Data Scientist DNA (Data Science Summit 2011)’, [online], Available: http://www.youtube.com/watch?v=bnZyHnQ00dA

Facebook, Facebook Developer Tools, [online], Available: http://developers.Facebook.com [15 March 2013]


Humbetov S. (2012), ‘Data-Intensive Computing with Map-Reduce and Hadoop’, Department of Computer Engineering, Qafqaz University, Baku, Azerbaijan


14. Appendix

AJAX Graphs

These are links for some AJAX Graph scripts that we can use for displaying the data visually in the project.

http://teethgrinder.co.uk/open-flash-chart/
https://developers.google.com/chart/
http://jpgraph.net
http://phpchart.net
http://code.google.com/p/birdeye/
http://phplot.sourceforge.net
http://www.flotcharts.org

Using Storm with Node

To use Storm inside a Node application, we need to use the following codes to setup a spout:

```javascript
var Spout = require('storm-node-multilang').Spout;
var sentenceEmitter = new Spout(function(events) {
    var collector = null;
    var seqId = 1;
    //You can listen to the spout "open", "ack" and "fail" events to
    events.on('open', function(c) {
        collector = c;
    });
    events.on('ack', function(messageId) {
        console.log(messageId + ' acked');
    });
    events.on('fail', function(messageId) {
        console.log(messageId + ' failed');
    });
    //The definition function must return a function used as
    //the nextTuple function.
    return function(cb) {
        var stream = T.stream('statuses/sample');
        stream.on('tweet', function (tweet) {
            collector.emit(tweet, seqId++);
            cb();
        });
    }, process.stdin, process.stdout);
    process.stdin.setEncoding('utf8');
    process.stdin.resume();
```
To setup the bolt, we just need to add this module and continue coding as shown above:

```javascript
var Bolt = require('storm-node-multilang').Bolt;
```

## Structure of Returned Queries

Here, I will provide an example of the results of various queries performed.

### Twitter:

```javascript
{
    created_at: 'Thu Aug 15 03:56:00 +0000 2013',
    id: 367857179295887360,
    id_str: '367857179295887360',
    text: 'Mi profe de inglés es bien flojo, ya empezamos bien :D',
    source: 'web',
    truncated: false,
    in_reply_to_status_id: null,
    in_reply_to_status_id_str: null,
    in_reply_to_user_id: null,
    in_reply_to_user_id_str: null,
    in_reply_to_screen_name: null,
    user:
        id: 764809826,
        id_str: '764809826',
        name: 'Toda La Vida ♥',
        screen_name: 'MadyChantes',
        location: '',
        url: null,
        description: '12/12/12♥ Mady&Luis..... Siempre Juntos, es una PROMESA♥♥',
        protected: false,
        followers_count: 135,
        friends_count: 120,
        listed_count: 0,
        created_at: 'Sat Aug 18 01:29:31 +0000 2012',
        favourites_count: 299,
        utc_offset: -18000,
        time_zone: 'Mexico City',
        geo_enabled: true,
        verified: false,
        statuses_count: 7429,
        lang: 'es',
        contributors_enabled: false,
        is_translator: false,
        profile_background_color: 'F1BFF5',
        profile_background_image_url:
            'http://a0.twimg.com/profile_background_images/37880000003206524/ff65dbd38dba2ccd757e270f2d6fe967.jpeg',
        profile_background_image_url_https:
            'https://si0.twimg.com/profile_background_images/37880000003206524/ff65dbd38dba2ccd757e270f2d6fe967.jpeg',
        profile_background_tile: true,
        profile_image_url:
            'http://a0.twimg.com/profile_images/378800000094296536/6b477b6dd216392669b54514cf9be74e_normal.jpeg',
        profile_image_url_https:
            'https://si0.twimg.com/profile_images/378800000094296536/6b477b6dd216392669b54514cf9be74e_normal.jpeg',
        profile_banner_url: 'https://pbs.twimg.com/profile_banners/764809825/1375674464',
        profile_link_color: 'D06BEC',
        profile_sidebar_border_color: '000000',
        profile_sidebar_fill_color: 'DDEEF6',
        profile_text_color: '333333',
        profile_use_background_image: true,
        default_profile: false,
        default_profile_image: false,
}```
WeatherMap:

```javascript
{ coord: { lon: 44.38, lat 33.3 },
sys: { country: 'IQ', sunrise: 1376274205, sunset 1376322685 },
weather: 
[ { id: 800,
  main: 'Clear',
description: 'Sky is Clear',
icon: 'Old',
iconUri: 'http://openweathermap.org/img/w/Old.png' ],
base: 'gdps stations',
main: 
{ temp: 313.25,
  pressure: 1002,
  humidity: 15,
temp_min: 312.15,
temp_max: 314.15 },
wind: { speed: 2.1, deg: 320 },
clouds: { all: 0 },
dt: 1376290800,
id: 98182,
name: 'Baghdad',
cod: 200 }
```

MapQuest:

```javascript
{ latLng: { lng: -42.812597, lat: 77.619235 },
adminArea4: '',
adminArea5Type: 'City',
adminArea4Type: 'County',
adminArea5: '',
street: '',
adminArea1: 'GL',
adminArea3: '',
type: 's',
displayLatLng: { lng: -42.812597, lat: 77.619235 },
linkId: 0,
postalCode: '',
sideOfStreet: 'N',
dragPoint: false,
adminArea1Type: 'Country',
geocodeQuality: 'COUNTRY',
geocodeQualityCode: 'A1XXX',
mapUrl: 'http://open.mapquestapi.com/staticmap/v4/getmap?type=map&size=225,160&pois=purple-1,77.6192349,-42.8125967,0,0|&center=77.6192349,-42.8125967&zoom=2&rand=-855416623',
adminArea3Type: 'State' }
```
RSS Parser:

```json
{
  type: 'rss',
  items: [
    {
      title: 'Oscar Pistorius due in court',
      summary: 'Olympic athlete Oscar Pistorius is expected to be served with an indictment in court next week following the completion of the investigation into the killing of his girlfriend, South African police said.',
      published_at: 1376483988000,
      time_ago: 'about 15 hours ago',
      guid: [Object],
    },
    {
      title: 'Aid group to leave Somalia over attacks',
      summary: 'Doctors Without Borders is pulling out of Somalia after more than two decades because of frequent attacks on its staff, the aid group said Wednesday.',
      published_at: 1376529024000,
      time_ago: 'about 2 hours ago',
      guid: [Object],
    },
    {
      title: 'Indian police hunt rapist of girl, 7',
      summary: 'Police say a 7-year-old girl was raped in an Indian train\'s toilet amid widespread outrage over sex crimes in the country.',
      published_at: 1376491118000,
      time_ago: 'about 13 hours ago',
      guid: [Object],
    }
  ],
  title: 'CNN.com - Top Stories',
  description: 'CNN.com delivers up-to-the-minute news and information on the latest top stories, weather, entertainment, politics and more.',
  ttl: '10'
}
```

**Import Data into MongoDB**

To load the lexicon list into MongoDB, I have used the following shell command:

```
mongoimport -d db_name -c collection_name
  -type csv -f fields_of_data
  --drop /location/csv
```

This adds the data from a CSV file to a new collection in MongoDB.

The commands to setup MongoDB on an AWS instance can be found at the following link: [http://docs.mongodb.org/ecosystem/platforms/amazon-ec2/](http://docs.mongodb.org/ecosystem/platforms/amazon-ec2/)
### Twitter4j Java Program

This is the Java program I created using Twitter4j to streams Tweets from twitter:

```java
import twitter4j.Twitter;
import twitter4j.TwitterFactory;
import twitter4j.conf.ConfigurationBuilder;
import java.io.*;
import java.util.*;
public class streamTwitter {
    static long score = 0;
    static Hashtable<String, Integer> lexicon = new Hashtable<String, Integer>();

    public static void main(String[] args) {
        ConfigurationBuilder cb = new ConfigurationBuilder();
        cb.setDebugEnabled(true);

        // BigDataProject Login
        cb.setOAuthConsumerKey("Fj4PyQOC1yExoB0gOiANg");
        cb.setOAuthConsumerSecret("XLVDDcIAjuuSepWk4amU6U0FWlgAgS2I5ZAHWpxKc");
        cb.setOAuthAccessToken("43315195-3f0r6OGY8ldZSaill9c3tvjblcxdhPnFoMPL2Ju");
        cb.setOAuthAccessTokenSecret("11tKeBq3dLWc16VrbzYdtq01kUAcavg8Yg");
        try {
            FileInputStream fis = new FileInputStream("resources/AFINN-111.txt");
            BufferedReader reader = new BufferedReader(new InputStreamReader(fis));
            String[] values;
            String line = reader.readLine();
            while (line != null) {
                values = line.split("\s+"),
                String word = values[0];
                int score = Integer.parseInt(values[1]);
                lexicon.put(word, score);
                // Proceed to read next line
                line = reader.readLine();
            }
            fis.close();
            reader.close();
            // System.out.println(lexicon.toString());
            catch (Exception e) {
                System.out.println(e.getMessage());
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```
//To change body of implemented methods use File | Settings | File Templates.
} 
@override
public void onStatus(Status status) {
    User user = status.getUser();

    //Get language of Tweet
    String language = status.getUser().getLang().toString();
    if (language.equalsIgnoreCase("en")){
        String username = status.getUser().getScreenName();
        System.out.println(username);
        String profileLocation = status.getUser().getLocation();
        System.out.println(profileLocation);
        String coordinates = status.getUser().getTimeZone();
        System.out.println(coordinates);
        System.out.println(status.getGeoLocation());
        long tweetId = status.getId();
        System.out.println(tweetId);
        String content = status.getText();
        System.out.println(content +"n");
        String[] stat=content.split("\s+\s+\s+\s+\s+");
        for (String word :stat){
            if(lexicon.containsKey(word)){
                score+=lexicon.get(word);
            }
        }
        System.out.println(score);
    }
    @Override
    public void onTrackLimitationNotice(int arg0) {
        // TODO Auto-generated method stub
    }
} 
FilterQuery fq = new FilterQuery();
String[] keywords = {};
fq.track(keywords);
tStream.addListener(listener);
tStream.sample();
}
Processing code

This is the processing code I created while learning to use processing. This creates two nodes that add a new arm as soon a tweet is received. There are two nodes one for Happy and Sad and the arms are added to the corresponding nodes. For now, the tweets are simulated using mouse clicks.

```java
float r=70;
float mX = 0;
float mY = 0;
PFont f;
float fs=10;
float lwidth = 0;
float rc=0, gc=0,bc=0;
float bc1x=-40;
float bc1y=-10;
float bc2x=30;
float bc2y=20;
float clx=r+50;
float cly=r+50;
boolean test=false;
int In=0;
void setup(){
  size(800,600);
  frameRate(30);
  f=createFont("Helvetica",14,true);
  smooth();
  noCursor();
}
void draw(){
  if (rc!=170&&&test==false)
  {rc++;
    if(bc>0)
    bc-=2;
    if(gc>30)
    gc--;
  }
  if (bc!=255&&gc!=127&&test==true)
  {gc++;bc++;if (rc!=0)rc-=2;}
  background(0);
  //background(rc,gc,bc,50);
  //Positive circle
  fill(255,0,0);
  stroke(255,255,255);
  if(r<150)
  r=ln;
  r=r+0.5 * sin(frameCount / 8);
```
Real-time Pictographic Representation of the Semantic Web – Dissertation Report

Heriot-Watt University, Dubai Campus
Ronny George Mathew, MSc. CSM, 2012

89

```java
clx=r+100;
cly=r+100;
// fs=fs+cos(frameCount/8);
//translate(mX,mY);
pushMatrix();
translate(200,200);
for (int i=0;i<ln;i++)
{
  bc2x+=0.5*sin(frameCount/4);
  bc2y+=0.5*cos(frameCount/4);
  bc1x+=0.5*cos(frameCount/4);
  bc1y+=0.5*sin(frameCount/4);
  //rect(0,0,2,70);
  fill(0,0,0,30);
  strokeWeight(.25);
  //noStroke();
  //curve(-30,-10,0,0,140,100,10,10);
  curve(bc1x,bc1y,0,0,clx,cly,bc1x,bc1y);
  fill(10,100,0,100);
  bezier(0,0,bc1x,bc1y,bc2x,bc2y,clx,cly);
  rotate(TWO_PI/ln);
}
fill(10,100,0,200);
strokeWeight(0.5);
stroke(255,255,255);

ellipse(0,0,r,r);
textFont(f,fs);
fill(255);

//Circle 2
//translate(mX+100,mY+50);
popMatrix();
pushMatrix();
translate(500,400);
for (int i=0;i<10;i++)
{
  bc2x+=0.5*sin(frameCount/4);
  bc2y+=0.5*cos(frameCount/4);
  bc1x+=0.5*cos(frameCount/4);
  bc1y+=0.5*sin(frameCount/4);
  //rect(0,0,2,70);
  fill(0,0,0,30);
  strokeWeight(0.1);
  //noStroke();
  //curve(-30,-10,0,0,140,100,10,10);
  curve(bc1x,bc1y,0,0,clx,cly,bc1x,bc1y);
  fill(100,10,0,100);
}
```
bez(x,0,0,0,bc1x,bc1y,bc2x,bc2y,clx,cly);
rotate(TWO_PI/10);
}
fill(200,10,10);
strokeWeight(0.5);
stroke(255,255,255);
ellipse(0,0,0,0);
//pushMatrix();
//translate(500,400);
for (int i=0;i<ln;i++)
{
rotate(0.1);
fill(255);
ellipse(i,0,10,10);
}
textFont(f,fs);
fill(255);
popMatrix();

//Smiley
fill(255,255,0,240);
stroke(0);
strokeWeight(1);
//translate(mX,mY);
ellipse(mX,mY, 100,100);
fill(255,255,255);
strokeWeight(1);
ellipse(mX-20, mY-20,20,20);
ellipse(mX+20, mY-20,20,20);
fill(0,0,0);
ellipse(mX-20, mY-20,10,10);
ellipse(mX+20, mY-20,10,10);

bez(mX-30,mY+15,mX,mY,mX+50,mY+50,mX+30,mY+15);
fill(255,0,0);
curve(mX,mY,mX-30,mY+17,mX+30,mY+17,mX,mY);

}

void mouseMoved(){
  mX=mouseX;
  mY=mouseY;
}
void mouseClicked(){
    textAlign(CENTER);
    if (test == false)
        test=true;
    else
        test=false;
    if(ln<150)
        ln++;
}

class Circle {
    float x,y; // location
    float dim; // dimension
    color c; // color
    float xSpeed;
    float ySpeed;

    Circle(float x, float y, float csize, color ccolor) {
        this.x = x;
        this.y = y;
        this.dim=csize;
        this.c=ccolor;
        xSpeed=random(-2,2);
        ySpeed=random(-2,2);
        //dim = random(20,50);
        //c = color(random(255));
    }

    void display() {
        float distance = dist(x,y,mouseX,mouseY); // distance between circle and mouse
        if (distance < 255) { // if distance is smaller than 255
            fill(255-distance);
        } else { // if distance is bigger than 255
            fill(c);
        }
        ellipse(x,y,dim,dim); // a circle at position xy
    }

    void update(){
        x+=xSpeed;
        y+=ySpeed;
    }
}
Mixing RSS Feeds in RSS MIX

RSS Mix provides a great service of combining multiple feeds into one. I have also evaluated MailChimp’s ChimpFeedr service and Yahoo Pipes. RSS Mix was the better option for my application. On the homepage of rssmix.com, can add multiple RSS feeds into a Text area and give it a name.

![Create a New RSS Mix](image)

Once this is done, we can click create! This will take us to another page with the link for our new feed in XML and JSON formats. We are using the xml format for our parser.

Adding Feeds...


Mix Created!

Your new RSS mix is here:

A JSON version of your new RSS mix is here:

Additional CSS

I have provided a copy of the CSS3 file I have created during the initial stages of development. I have used various transition effects of CSS in this file.
School of Mathematical and Computer Sciences

Ethics Approval

Student Projects

Title of project: Real-time Pictographic Representation of the Semantic Web

Supervisor: Talat Shaikh

Purpose of study: Analyse real-time data available on the internet and to find interesting information from this data.

Are human subjects or personal data identifiable with living people involved?

Yes

No

If NO sign and date the form, otherwise answer the rest of these questions

Is the project mainly concerned with other matters and the only ethical consideration the use of human subjects in interface evaluation?

Yes

No

If YES and the subjects are staff of students of Heriot-Watt University, no ethical approval is required, except for the requirement for subjects to sign the standard consent forms and to store these consent forms. Data must be anonymised.

If NO, then you must complete the full research proposal ethical approval form.
Please sign the following:

I, the student undertaking the project, certify that this is a true reflection of the intended study, and that I will seek the necessary approval if the nature of the project changes.

Name (please print)  RONNY GEORGE MATHEW
Signature
Date  15th August 2013

I (as the project student's supervisor) have checked the above for accuracy and am satisfied the information provided is a true reflection of the intended study.

Name (please print)  TALAL SHAIKH
Signature
Date  15th August 2013
## MACS Risk Assessment Form (Project)

### Student:  RONNY GEORGE MATHEW  

### Project Title:  REAL-TIME PICTOGRAPHIC REPRESENTATION OF THE SEMANTIC WEB  

### Supervisor:  TALAL SHAIKH  

### Risks:  NONE  

<table>
<thead>
<tr>
<th>Risk</th>
<th>Present (give details) (tick if present)</th>
<th>Control Measures and/or Protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Office environment- includes purely software projects</td>
<td>![Checkmark]</td>
<td>Nothing</td>
</tr>
</tbody>
</table>
| Unusual peripherals  
e.g. Robot, VR helmet, haptic device, etc. | | |
| Unusual Output  
e.g. Laser, loud noises, flashing lights etc. | | |
| Other risks | | |