Design and Implementation of a News Reader based on Social Networks

Abhinav Uppal  
Sandeep Kumar Gupta  
Sahil Singla

Advisor: Dr. Aaditeshwar Seth  
Department of Computer Science and Engineering,  
IIT Delhi.

II Semester 2010-11  
BTech Project CSD 412
Contents

1 Introduction 3
  1.1 Related Work . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
  1.2 Overview of the System . . . . . . . . . . . . . . . . . . . . . 4
  1.3 Analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

2 The System 5
  2.1 Central Server . . . . . . . . . . . . . . . . . . . . . . . . . . 6
    2.1.1 Facebook Application . . . . . . . . . . . . . . . . . . . . 6
    2.1.2 Clustering Overview . . . . . . . . . . . . . . . . . . . . 7
    2.1.3 Database . . . . . . . . . . . . . . . . . . . . . . . . . . 7
    2.1.4 Servlets . . . . . . . . . . . . . . . . . . . . . . . . . . 9
  2.2 Client . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
    2.2.1 Protocols . . . . . . . . . . . . . . . . . . . . . . . . . . 12

3 Clustering 13
  3.1 Limitations of modularity based clustering . . . . . . . . . . . 16

4 Visualization 16
  4.1 Color Coding for Topics . . . . . . . . . . . . . . . . . . . . . 16

5 Analysis and Results 17
  5.1 Twitter Dataset . . . . . . . . . . . . . . . . . . . . . . . . . . 17

6 Future Work 18
1 Introduction

There is a lot of content available on the Internet today in the form of news, articles and blog posts. It is becoming increasingly difficult for users to find content over the Internet that is relevant to them or that interests them. Another discernible trend is the steep rise of Online Social Networks (OSNs) over the last few years.

There has been a marked shift in the way people are using the Internet. Historically, the Internet was based on a Producer-Consumer content model supporting one-way broadcasting and dissemination of information. Today, OSNs such as Facebook and Twitter are driving new forms of social interaction. They have enabled users to actively contribute to the content available on the Internet by sharing posts, adding comments and participating in online activities. One of the most popular online social networking sites, Facebook, currently has more than 600 million active users. An average user on Facebook has 130 friends and people spend over 700 billion minutes per month on Facebook. These figures indicate that OSNs have a potential to revolutionize the Internet experience for users.

In this report we describe the design and implementation of a system that provides a new way to read news by leveraging the connectivity in Online Social Networks (OSNs) making it easier for a user to find news content that interests him. With more and more people interacting through OSNs, there is tremendous potential to tap into. The OSNs are transforming online user behaviour in terms of users initial entry point, search, browsing and purchasing behaviour. Congruent to our beliefs, some experts suggest that social media will become the Internet's new search function - predicting that people will spend less time navigating the Internet independently and instead search for information or make decisions based on word-of-mouth recommendations from their friends.[8]

1.1 Related Work

There has been a considerable interest in providing selective news content to users using their contextual information. Various projects have been undertaken by the commercial companies R&D teams to study the reading patterns of online users and come up with better systems.

Project Cascade by New York Times R&D [5] is trying to come up with a picture of how information propagates through the social media space.
Through a visualization tool, it plots the timeline showing various events in the history of an article thus studying the viral propagation and sharing habits of the users.

**The Conversation Cloud by Economist Online** [6] compiles users opinions on various topics from comments posted on articles, blogs, and debates to feature the most popular discussions on the site. Their visualization tool allows the users to navigate through a cloud of comments, opinions enabling an individual user to look for the the conversation of his interest.

**Google Reader** [7] is a web-based aggregator that lets you subscribe to the top news services and also follow the people you like. Users share the news items, articles which they like and the shared content appears in the news feed of their followers. It also provides customized news suggestions to an individual. However, the list of followers and the shared news content are not visualized through a tool and user is left to read a river of news. Also we are focusing on aggregating news at a cluster level instead of the user level thereby ensuring that it is not revealed what an individual user has read and hence maintaining their original reading habits.

We believe that we are the first ones who are trying to visualize the shared news content on a social network graph thus benefiting from the connectedness and similar interests of users in a strongly connected cluster.

### 1.2 Overview of the System

Our system exposes online readers to news read by different social groups by identifying different clusters in the users OSN. Users of our system first register with a Facebook application that fetches friend lists of these users. We build a friendship graph for all the users that register with our system. This graph has its nodes as the registered users and any two nodes are connected by an edge if the two corresponding users are friends on Facebook. We cluster this friendship graph into a hierarchical structure (dendrogram) of small social groups by running a variant of the Louvain Algorithm. The entire social network is represented by the root node which has a group of clusters as its children, which further divide into groups of clusters. All the clusters are leaves of this dendrogram. Two clusters are siblings if they are more connected to each other than they are to other clusters.

Users also install a Firefox extension when they register with the Facebook application. This Firefox extension detects when users read news online. The
extension sends the news article on the page to a free to use service called Open Calais to fetch social tags, topics and entities associated with the news item. It then reports the link to our system. Based on these tags our system suggests related news to this user from the cluster that he belongs to or the clusters that he has subscribed to.

We also provide an interactive visualization of the hierarchical structure of the OSN allowing users to zoom into particular clusters of interest for more news. When a user hovers his mouse pointer over a particular cluster the most popular news items in that cluster are displayed to him. Users can subscribe to clusters of their interest. Additionally, users can search for social tags, personalities, places, topics etc. and determine which clusters are talking most actively about that search query.

Users of our system cannot tell which other users are reading what. They can only know that some user in a particular cluster has read some news item. Additionally, we provide an incognito mode for users that they can turn on to stop sharing content with our system.

1.3 Analysis

We have created a data set of topics, social tags and entities related to tweets in a twitter database collected by Munmun De Choudhury et al [11] by sending these tweets to Open Calais. We also used the follow links on Twitter to create a follower graph where two users are connected if they follow each other. We then clustered this graph into communities. We analysed how frequently different communities talk about various topics, social tags and entities. We also analysed the effectiveness of Open Calais for parsing such tweets/status messages of users.

The rest of this report is arranged in the following way:

2 The System

We have developed a prototype of our news reading system using a central server for communicating with all client nodes. The main components of our system are a central server, client nodes and a free to use web service that identifies keywords from news items.
2.1 Central Server

A central server is used to communicate with all the client nodes in the system. A schema of the functionalities of the server is given below:

2.1.1 Facebook Application

Facebook is the most used social networking website today with more than 600 million active users around the globe[9]. Facebook provides a rich API that lets the app developers access a user's publicly available information into their applications. More and more independent websites are trying to enhance their visitors experience by integrating various features of Facebook and according to [10], over 20 million Facebook applications are being installed everyday.

For our project, we made a Facebook application (http://apps.facebook.com/iitnewsreader/). Through this application, we fetch the users list of friends using the Facebook API. We store the Facebook IDs of the registered users along with their list of friends in a database on our central server. This
Figure 2: Functionalities of the central server

information is further used to build and store a friendship graph. In the
graph, all nodes represent clients and an edge between two nodes represents
a friendship link between the two clients on Facebook.

2.1.2 Clustering Overview

The friendship graph that we made from the data collected by the facebook
application is then divided into clusters using a clustering algorithm. We use
a variant of the Louvain Algorithm to cluster the sampled social network.
Each cluster is given a unique ID. A mapping from registered users to their
cluster IDs is maintained in a database on the server.

2.1.3 Database

The database on the central server keeps track of the URLs and tags for
news items that have been read by the users of our system. It also maintains
the number of times a news item has been read in different clusters. It also
maintains cluster IDs of all client nodes. There are seven tables on the server:

1. Users (T1): Maintains a mapping from users who have registered with
our Facebook application to their respective cluster IDs.
2. **Friends** (T2): Maintains the friendship links between different users who have registered with our Facebook application.

3. **NewsTags** (T3): This table contains a list of social tags associated with each news item that has been read by the users of our system.

4. **ReadCount** (T4): For each news item (url) being read, we maintain a count of how many times it has been read in a particular cluster (having its cluster ID = cid). It also contains the latest time when the particular news item was read in that cluster.

5. **Tweet_topics** (T5): This table contains the data set that we have created using a Twitter Database collected by Munmun De Choudhury et al. [11]. It contains a list of usernames on Twitter and the topics identified by their tweets.

6. **Tweet_tags** (T6): It contains a list of usernames on Twitter and the social tags identified by their tweets.

7. **Tweet_entities** (T7): It contains a list of usernames on Twitter and the entities identified by their tweets.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Name</td>
<td>FB ID</td>
</tr>
<tr>
<td>T2</td>
<td>Friend1</td>
<td>Friend2</td>
</tr>
<tr>
<td>T3</td>
<td>URL</td>
<td>Social Tags</td>
</tr>
<tr>
<td>T4</td>
<td>URL</td>
<td>Count</td>
</tr>
<tr>
<td>T5</td>
<td>Username</td>
<td>Tag</td>
</tr>
<tr>
<td>T6</td>
<td>Username</td>
<td>Topic</td>
</tr>
<tr>
<td>T7</td>
<td>Username</td>
<td>Entity Type</td>
</tr>
</tbody>
</table>

Figure 3: Database Structure
2.1.4 Servlets

The Firefox extension sitting on our clients communicates with the central server using HTTP post requests. We have created 4 Java Servlets to cater to these request made by the Firefox extension:

1. **GetTime**: Returns the server time to the client for synchronization.

2. **UpdateNews**: Takes a URL and a serialized list of associated keywords as arguments and stores them in table T3 of the database on the central server.

3. **GetNews**: Takes cluster ID and a timestamp as arguments and returns all the news items read in that particular cluster after the given time as an XML document.

4. **GetColorsForTopic**: Takes a query string as argument and returns the number of times twitter tweets related to that query string have been read in different clusters.

2.2 Client

Clients are required to install a Firefox extension when they first register with the Facebook application. Firefox is one of the most popular browsers used to browse the Internet and has a very powerful development framework. For our project we built a Firefox extension that sits at the client end and communicates with the central server.

The Firefox extension maintains a local database at the client node. This database acts as a cache. For this purpose we use a very light database (supported by Firefox) that supports sql type queries (sqlite). The local database has the following structure:

1. **ReadNews** (T1): This table contains the URLs of news items that have been read by nodes in the users cluster, along with a count of unique readers in the cluster that have read the news item. A boolean variable read has also been saved along with the news URL to indicate whether it is this user who has read this news or someone else in his cluster. This helps us to display only those items in the ticker which have been read by others.

2. **OtherNews** (T2): This table acts as a cache for news items belonging to other clusters. It stores the URLs and maps them to the cluster IDs that they have been read into along with a read count.
3. **NewsTags** (T3): This table maps news items to the social tags and entities that have been identified by Open Calais. All the news items from the users cluster and users subscribed clusters are present here. For other clusters, the news items are cached into the table.

<table>
<thead>
<tr>
<th>T1</th>
<th>URL</th>
<th>Readcount</th>
<th>Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2</td>
<td>URL</td>
<td>Readcount</td>
<td>Cluster ID</td>
</tr>
<tr>
<td>T3</td>
<td>URL</td>
<td>Social Tags</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 4: Local Database Structure**

The Firefox extension performs the following functions:

1. **Detect news items being read by the user:**
   The extension detects whenever a new page is loaded into the browser. The extension takes care of multiple browser windows and tabs. Realising that this functionality is not well documented on the web, we have documented how to handle multiple tabbed browsing environments. Please visit the Project Wiki for the same. Whenever a url is loaded, the URL is parsed by the extension. If the domain name of the web page matches one of the news providers that we are sampling, we parse the URL and the html page according to the website standards to fetch the News Headline, the article and the web link.

2. **Fetch social tags and entities for the news item:**
   Once a news item is detected and if the user is not browsing in incognito mode, we send the text in the news article to Open Calais. We receive an XML document in response from Open Calais. The response is then parsed to retrieve the social tags and entities associated with the news item.

3. **Add read URLs to local database (Cache):**
   The URL of a read news article is added to the local database in table readnews with read = 1 if it is not already present in the table. If the news article is already in the table and read = 1 then nothing is done. If read = 0, then the news items readcount is incremented and its read is set to 1. The list of social tags and entities identified by Open Calais is added to the table newstags.
4. **Share news items with the central server:**
   Whenever a news item is read, its contents are reported to the central server.

5. **Update local news database:**
   The Firefox extension synchronizes its time with the server and remembers the last time it had updated its local news database. Each node pings the central server for updates in its own cluster. For this, it sends lastupdatetime from its configuration file to the server, and asks Are there any more news after this time?. The server then selectively sends back all the popular news items that were read in the nodes cluster post this update time.

   Actions taken on getting response from server:
   (a) Any news item which a user has not read is added into its database.
   (b) Already read news items readcount is updated.

6. **Provide related News Items:**
   When a news item is detected by the extension and its keywords are retrieved from Open Calais, the extension searches the local database for news items with matching keywords. These news items are then displayed in a horizontal ticker at the bottom of the news page. Users can click on these buttons and visit these webpages to read related news.
2.2.1 Protocols

1. Synchronization of time with the server

   ![Diagram showing protocol flow]

   Server:
   - 1. Ping Time
   - 2. Time in ms

   Client:
   - 0. Store local time
   - For rest of session:
     - Server time
     - Current local time
   - For time from server

2. On new news item being read

   ![Diagram showing protocol flow]

   1. Plugin detects a news item being read
   2. Send URL
   3. Keywords
   4. Send URL and Keywords and Cluster ID
   5. If news item exists then increase count else add new entry.
3. Periodic updates

3 Clustering

We studied the following three algorithms: CNM [13], Wakita [14] and Louvain Algorithm [15]. All of them are greedy algorithms using modularity as the metric for optimization.

\[
\text{Modularity } Q = \sum (e_{ii} - (a_i)^2)
\]

Where:
- \(e_{ii}\): ratio of number of links between nodes in community \(i\) and the total links in the graph.
- \(a_i\): ratio of number of links crossing the boundary of community \(i\) and the total links in the graph.

Modularity ranges between -1 and 1 and modularity close to 0 means that edges in a community are almost completely random.

Here is a brief description of the algorithms.

1. **CNM[13]**: This is a hierarchical agglomeration algorithm for detecting community structure. It is a bottom-up greedy algorithm to maximize modularity. As compared to its predecessors, it uses sophisticated data structures to bring the running time to \(O(m.d.\log n)\). Here \(m\) is the number of edges, \(n\) is the number of nodes and \(d\) is the number of levels of the dendrogram. Beginning with each node as a different community, in each iteration it merges two communities such that \(\Delta(Q)\) (change in modularity) is maximum. It stops when \(\Delta(Q)\) becomes negative for every possible merging.

13
2. **Wakita**[14]: Just like CNM algorithm, this also tries to greedily maximize total modularity. This algorithm also tries to prevent unbalanced growth. Problem with other algorithms prior to it was that they resulted in some very large clusters and other clusters being very small. Wakita’s algorithm addresses this problem by also taking into consideration the current size of the cluster. Thus it discourages large clusters to grow further. This algorithm is scalable to more than 1 million nodes.

3. **Louvain**[15]: This algorithm uses the idea of consolidation whereby clusters of similar sizes are encouraged to merge. Also in each iteration, running in two phases, it merges all nodes and clusters that maximizes delta(Q). Thus both modularity and the running time is improved greatly. In the first phase, the algorithm moves nodes from one cluster to its neighboring clusters so as to maximize delta(Q). The process is continued till no further improvement is possible. Now in the second phase, the algorithm replaces clusters with nodes, give sum of weights of nodes as new link weights and returns to phase 1.

   Based on results from [1], we arrived at the conclusion that Louvain’s algorithm is most promising and relevant for our project. The reasons are mentioned below:

   1. Louvain’s algorithm tries to ensure balanced growth, which makes it better than CNM.
   2. It can run on more than 10 million nodes.
   3. The algorithm runs recursively and merges some clusters in every iteration. Hence the output is in the form of a dendrogram.
   4. Running visualization algorithms is far simpler on dendrograms. Also its structure can be easily modified for a better visualization.
   5. Modifying the output of the algorithm so as to control community sizes is easier to implement in a tree structure.
   6. The source code is easily available on their website.

The algorithm was tested on a facebook social network data set of 63,731 nodes [4]. We are thankful to Alan Mislove of Northeastern University for cooperating with us in this regard. We realized that there were clusters of size greater than 14K nodes in the base level. As such sizes are not desirable for our news feed system, we decided to re-run the Louvain algorithm on the
sub-graph induced by the vertex set of a large cluster.

To break large communities, we wrote a C program that finds communities of size larger than MAX nodes, re-runs the algorithm and replaces the old community with the dendrogram produced by the algorithm. Re-running the clustering algorithm does not disturb other communities. As the new clusters are attached to the same node that contained them before fragmentation, its position in the hierarchy is preserved. The running time of the complete program was still less than 10s. It produces an XML file in the format that can be directly used in our visualization tool.

The current modifications have managed to control large cluster sizes but visualizing them has revealed that clusters of very small sizes are still possible. As users in such clusters will not be connected to many other friends, news feed system will not be very efficient in suggesting them related news. Next, we plan to merge communities these extremely small communities by either merging them into a single new community or by adding them into a neighboring large community.

Following figures reveal the differences that were observed in the visualization tool.

![Figure 5: Trace1](image1.png)

![Figure 6: Trace2](image2.png)

While Fig. 5 shows original large clusters, Fig. 6 shows how these large clusters were broken down to obtain smaller clusters.
3.1 Limitations of modularity based clustering

Although modularity maximization methods are very effective in community detection but still they suffer from some limitations.

1. There is a resolution limit in modularity based clustering. This means that the algorithms cannot detect communities smaller than a certain scale.

2. Overlapping communities cannot be detected using modularity maximization. This is a serious limitation as in OSN, most people are part of many clusters and a model handling overlapping communities will be helpful. Palla [8] has tried addressing this problem but in our present model, we are not addressing this limitation.

4 Visualization

4.1 Color Coding for Topics

As already talked in the paper, some experts suggest that social media will become the Internet's new search function - predicting that people will spend less time navigating the Internet independently and instead search for information or make decisions based on word-of-mouth recommendations from their friends.[8]

We strongly believe in this statement and have included its provision in our visualization technique. We are associating all the past read news articles with Topics, Social Tags, Entities (We shall be using keyword as a single term to refer to all these). Certain clusters on the social network are more active in reading about new content related to one particular topic while certain other clusters have different reading habits. For a particular keyword we calculate, on the fly, which are the top clusters that are reading news articles related to this keyword. We then color code these clusters assigning a darker shade to the clusters that are more active as compared to others. Users can then subscribe to these clusters to continuously get updates related to this topic from this cluster. This method provides a new way to keep yourself updated with a particular topic since the vast news content has already been filtered by a large number of people.
5 Analysis and Results

5.1 Twitter Dataset

We wanted to check our scheme of color coding on a real dataset. For this, we got a Twitter dataset from Munmun De Choudhury [11]. It contains two file:

1. socialgraph.data: Contains the friendship links between users on Twitter. Only those links are included in which both the users are following each other.

2. tweets.data: This file contains a list of tweets by different users sorted according to the time. The format of lines in this file is:

   Username Date Time Tweet

We present here only the statistics of the dataset, for a detailed analysis on how the data set was collected through focused crawling please refer to the paper [11].

| #nodes | 465,107 |
| #edges | 836,541 |
| #tweets | 25,378,846 |
| Time Span of tweets posting times | Oct 2006 - Nov 2009 |

We took a subset of 139,600 tweets from the tweets.data and sent a request to Open Calais for each one of them. We got 34,901 replies back with a Topic for each of them showing that only about 1 out of every 4 tweets in the original dataset were actually meaningful tweets from the perspective of Open Calais. We selected the users who had posted these meaningful tweets and found out they were 3320 in number. We looked in the socialgraph.data to check the friendship links of these 3320 users. So, effectively, we truncated the original dataset to the following table:

| #nodes | 3320 |
| #edges | 9132 |
| #tweets | 139,600 |
| #meaningful tweets | 34,901 |

Note that an edge exists in our truncated data set only if one of its nodes of
that edge is among the 3320 users. We ran our variant of Louvain algorithm on these users and clustered these users into small social groups as shown below. Open Calais divided these 34,901 tweets into 18 broad categories based on their topics. The topics ranged from Social Issues, Law Crime to Sports. We used our color coding scheme for these topics and got the following results:

![Figure 7: Visualization of the Clusters](image)

In the above figure, the figure on the left hand side shows default view of the clusters of Twitter dataset. There are many one user clusters towards the circumference because of the power law in the degrees of users in twitter. Many users just follow powerful celebrities and are not followed at all in Twitter Online Social Network. We then chose one topic *Technology, Internet* and use it to color code all the clusters. The clusters which were *talking* more about these topics have been marked with darker shades of blue as compared to the clusters who were talking less about this topic. Now an interested user can subscribe to a darkly colored cluster and follow all the subsequent news items being read by users of that cluster.

### 6 Future Work

Since we are done with the prototype implementation of the system, we will be deploying it in IIT to study users response towards our system. The statistics we are interested in collecting are as follows:
1. Create a database of the news articles being read by different users. Currently all our analysis has been on twitter or facebook datasets. A news articles database will give us a real world example providing insights into the reading habits of the users.

2. There are now three ways to read articles:

   (a) Read them independently on a news content provider.
   (b) Click on one of the related news article being shown in the status bar.
   (c) Go to the visualization tool, and read an article from one of the clusters.

   We are interested in knowing which is the most prominent means by which a user is reading news content. This would enable us to evaluate the impact of our system on the way people are reading news.

Explore other possible ways to visualize the shared news articles on a social network. Our technique provides a good way to search for a strongly connected component in the social graph that is active in conversations about a particular topic. It will be interesting to visualize how news disseminate from this strongly connected component. This would lead to discovery of other clusters having similar interests as this cluster. Possible ways of showing this are stronger edges based between clusters based on the past history of news propagation.

*Topic Based Clustering* A dynamic clustering based on the topics. User is allowed to chose a topic and the nodes get dynamically rearranged to form new clusters based on that topic. This would require to keep track of the past reading history of all the nodes and keeping metrics which would help to find users more interested in particular topic as compared to others.

*Distributed Implementation* A central server model was only a prototype acting as a proof of concept for this idea. The original idea was to implement it in a completely decentralized manner. We have developed a strong theoretical framework and after a test deployment, we will start to work on a distributed implementation of the system.

*Personalizing Clusters* One way to memorize clusters on a Social Network Graph is by their relative position to other clusters. However, if the social network rapidly changes then it would be difficult to keep track of relevant
clusters. One could potentially assign meaningful labels to each cluster such as geographical locations. Facebook also provides location of the user in its publically available API. If large number of users belonging to a particular clusters also share their geographical location, then that cluster can be assigned a label corresponding to this location. This will also lead to an additional feature of reading what people are interested in a particular geographical area.

References


