UNIVERSITÁ DI PISA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea Magistrale in Computer Engineering

Mining Twitter: Graph Analysis of Interactions among Users

Tesi di Laurea

Candidato:
Daniele Battista

Relatori:
Prof. Marco Avvenuti
Prof. Maurizio Tesconi
Ing. Fabio Del Vigna
Dott. Salvatore Bellomo

Anno Accademico 2015/2016

Abstract

Starting from early 2000s, social network websites became very popular; these social media allow users to interact and share content using social links. Users of these platforms often have the possibility to establish hundreds or thousands of social links with other users. While initial studies have focused on social networks topology, a natural and important aspect of networks has been neglected: the focus on user interactions. These links can be monitored to generate knowledge on said users as well as their relationships with others. There has been, lately, an increasing interest on examining the *activity network* - network able to provide, once traversed, the actual user interactions rather than friendships links - to filter and mine patterns or communities.

The goal of this work is to exploit the Twitter traffic in order to analyze the users interactions. In order to do so, our work models tweets posted by users as activities list in a graph called *activity network*. Then, we traverse it looking for *direct* (e.g. mentions by user, retweets, direct replies etc.) and *indirect* (list of users mentioned in a tweet, users retwitting the same tweet produced by another user, etc.) relationships among users in order to create the *users interactions graph*. We provide a weight schema by which assign a value to interactions found. The obtained graph shows the connections among users and, thanks to their weighted links, those users who have stronger links, such as *Verified Accounts* or *"propaganda users"* or *cliques of users*, clusters of users interacting with each other. Those entities may be interesting to investigate in several fields like Open Source Intelligence or Business Intelligence. This work has been developed on a distributed infrastructure able to perform these tasks efficiently.

The network analysis leads to some considerations: firstly, it is necessary to identify all meaningful interactions among users, which typically depend from the social network and the activities performed. Secondly, many nodes (profiles) with high indegree are associated to mass media and famous people, and thus a filtering phase is a crucial step. Finally, it is remarkable to see that experiments carried out at different moments could lead to

very different results since many similar topics may not involve the same users in different moments.

This work will describe the state-of-the-art of the network analysis, and will introduce the architectural design of the system, as well as the analysis performed with the challenges encountered. Results collected by our analysis lead us to the conclusion that, despite being in its preliminary stages, focusing on social interactions is important because it may reveal connection of particular users willing to perform actual activities which may gain interest in intelligence organizations.

Contents

List of Figures

List of Tables

1. Introduction

The world around Social Networks is becoming more and more wide since it is possible to interact with them in so many different ways and users' lives are becoming more and more conditioned by them. Through Social Networks people can have their own pages in which they can share content with friends or other registered people. Activities on social media are generally related to content a user shares or produces, which can be considered one of the most important entity in a Social Media. It is called Post in Facebook and Instagram, Status in Twitter or Video in Youtube but, after all, it is just the fulcrum of a Social Network. The importance of a Post has to be seen in all the elements that are contained in it when it is sent, in a sort of way, to the Social Network. Each post, in the most general case, can contain text, expressing the thoughts of the user, a media, which can be a photo or a video, a link to a website, showing a content the user wanted to share with the community, but also mentions to other users, involving them in the post, or *hashtag*, a short word marked by the # symbol as to indicate a sort of topic of the post. All this elements can be a valid support to data scientists for inferring new information, through the adoption of information retrieval and analytical techniques.

Starting from the above mentioned aspects and given also the extensiveness of social platforms, it is straightforward for a data scientist to start exploiting Social Media for research and business. As stated in [1], in January 2009 Facebook had more than 175 million of active users, every minute 10 hours of content were uploaded to Youtube, Flickr gave access to over 3 billion photographs. If we try to look for some statistics about social media nowadays, we can observe that the current number of active users on Facebook is 1.65 billion, 400 million on Instagram, 320 million on Twitter, over 1 billion for Youtube. Other statistics show that Facebook has 8 billion average daily video views from 500 million users, it adds 500,000 new users every day (6 per second) and the average number of friends is of more than 300. On Twitter, instead, the average number of followers is of about 200, while 500 million Tweets are sent every day (which means 6,000 Tweets every second). In Youtube the number of hours of video uploaded every minute passed from 10 to 300, and the actual number of languages

covered is 76, comprising the 95% of the Internet Population. Instagram has 80 million new photos uploaded every day and more than 40 billion photos shared in the same period [2]. These numbers highlight how much the social media phenomenon exploded in a short time and how huge volumes of data are when we talk about these platforms and websites.

The enormous quantity of data that can be retrieved from social media is characterized by the presence of noise, that is not meaningful information that can mislead the data analysis. To address the problem it is necessary to adopt filters able to detect contents not suitable for the defined purposes. Consequently, social media content must be carefully managed using proper tools and techniques, keeping in mind that data is typically affected by noise and must be handled with care.

1.1 Social Media Content Motivations

Social media provide a large quantity of data that can be retrieved and used to extract important features which could reveal some issues and relationships in data. Indeed, it is worth to ask ourselves what kind of data can be retrieved, and how the structure and API of social media ease this task. Furthermore, relationships among users usually convey a lot of information about the way they interact and the content they share, providing interesting insight in communities that share the very same interests, and in which the members might influence each other.

The answer proposed by [3] suggests how a social media is organized. For a full comprehension, it is necessary to consider all these aspects:

- Identity
- Conversations
- Sharing
- Presence
- Relationships
- Reputation
- Groups

Generally a social media provides a sort of template in which a user can specify all the personal information like name, surname, city and age. These information are part of the

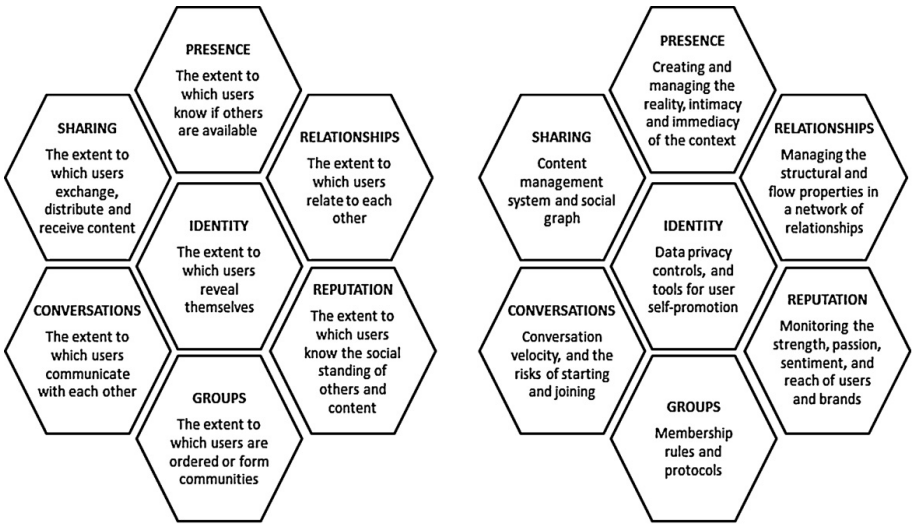

Figure 1.1: *Social media functionalities & implications*

Identity of the user on the social media. This is an important feature that identifies users and allows an initial segmentation of the subscribers.

Since users typically interact with each other, **Conversations**, namely the exchange of direct messages, represent the simplest kind of link among users. Another way to express Conversations is considering a collection of comments and replies, since two or more users are answering each other on a particular topic. This means that they are interested in that topic and could be joined in a group.

The **Sharing** functionality, similar to Conversations, allow users to propagate content produced by other users, suggesting a common interest among them. This also means that probably, in the future, they will repeat the same action on other content.

The **Presence** is a functionality that summarizes the location of a user at a certain time or at a specific event. Users who are located in a certain place could be grouped specifying that their interests are probably equal, but also, showing their presence just in that place, could convince other users to join them there augmenting the importance of the relationships among them. It is the typical case of Foursquare [1], a social network in which users can checkout the place they visit and share their position with others.

Relationships are probably the most important functionality in a social media because join users and state that there exist a link among them, even if they never interact. Relationships are modelled in each social media and could have a strong or a weak importance. Some examples of this are in Twitter and LinkedIn. In the former, relationships are asym-

[1] https://it.foursquare.com

metric and represented by friend/follower labels; in the latter, instead, they are symmetric and they have a strong meaning saying that two people are connected because they work in the same field or collaborate. Clearly, depending on the social media, it is possible to assign a different weight to each relationship. On Twitter, the power of a relationship has not a strong relevance since a user could be interested in what another user says or shares, but the interest could not be reciprocated. Instead, LinkedIn and Facebook provide a sort of control on relationships in the sense that a user has to accept a request of relationship before joining to another user: this reciprocal acknowledgement enhances the strength of the link. Social behaviors have the power to strengthen a relationship depending on how two users interact in their real life. If two users are both members of a particular club or work for the same company in real life, typically their interaction will be mirrored in social media.

Relationships are the foundation for the **Groups** functionality. Social media invented this construct with the only aim to cluster users, having the possibility to propose them more suitable solutions. Social media are able to use this information to propose pertinent user generated content members and to suggest more targeted advertisement. Groups can be open or secret, could require to be accepted by administrators to enter, or could require active participation. Moreover, due to the large traffic of data on social media, Groups have a particular interest because they produce important information about characteristics of users involved in and create a very precise interaction map.

The last functionality to be analyzed is **Reputation**, a characteristic showing the confidence a user possesses. Reputation in social media can be modelled on people but also on what is said by them. Many social media have verified profiles used by celebrities, sportsmen, singers, etc., to interact with the crowd. But typically crowds do not have such possibility, and users have to build their Reputation through contents endorsed by the others. For this reason, social media need a mechanism to evaluate the importance of posts, identifying people who are able to state influence a topic. Techniques that strive for achieving this are based on characteristics like strength (number of times a user is mentioned), sentiment (ratio between positive and negative mentions), passion (number of times a user is mentioned by others) and reach (number of users talking about a user divided by number of mentions). This mechanism is generally used by users themselves who select their friends and relationships through a personal evaluation of the content in each post.

The key characteristics of the social media world presented above should lead to understand why it's important to get and use data coming from social media. This data can be used in several fields and applications depending on what is the real goal to achieve. Enterprise may use them to control the marketing trend of a new item, Police may use it to monitor the

traffic or the general situation in a crowded square and so on. Data from Social Media are available through the related API that have to be called in order to get what we need.

1.2 Users Interactions Graph and SNA

The ability to work on data coming from Social Media has to be seen when we are going to apply Social Network Analysis Techniques on a network in charge for representing a graph of interactions among some entities, which can preferably be users. The goal of SNA is to learn socially from peers available in the network, peers that are connected themselves through relationships and interactions established in a way the modeller decided in a former phase. Networks like these admit to track and monitor some users or some behaviours understanding the role a user has or explaining the reason a particular connection.

In past Social Network Analysis was considered as divided in two main groups: Social Learning and SNA. Social Learning concerns with the possibility to learn and create something new thanks to interactions with other peers. In fact we have to cite the notion of *zone of proximal development*, which states that knowledge situated within peers can top up the zone of proximal development. Starting from this concepts, the concept of *Constructivism* and *Connectivism* took their part in the discussion: the former describes learning as a process of constructing new ideas and concepts from known concepts, by an individual learner or by several learners collaborating and states that knowledge can be spread on and through social interactions; the latter defines learning as a process of connecting specialised nodes or information resources, as an ability to see connections between fields, as ideas and concepts. The evolution lead to the proposal of the Social Network Analysis Applications which, first of all, need to collect data to apply related techniques. Ways to collect data can be seen in *sociocentric networks* which focus on the interactions between people on a network level, studying structural patterns of interactions in a network with the specific aim to generalize them to other networks. An example can be seen in monitoring real-world data of human interactions, such as Twitter followers and retweets, Facebook contacts, email traffic and so on. Another way is represented by *egocentric networks*, which focus on the networks of individuals. These networks can be constructed asking one or more participants to identify their contacts and asking also to those contacts if they are related or not to the participants. They can be constructed also through the *snowball method*, asking a participant for contacts, then asking to those contacts for their contacts ad so on until a stopping criterion is met.

Once these networks have been created, it is possible to study them at three different levels: the most general level , in which we analyze the whole network, the number of links,

the density etc.; the community level, in which we would like to discover the number of subnetworks within a network, the so-called connected components, which have weaker links between them than among their contained nodes and arise themselves thanks to shared interests of members; the individual level, in which we would like to discover the centrality of an individual in the network, characteristic able to discover to what extent someone is "needed" in terms of information flow (e.g. how often is someone in-between two other people?). The applications, as specified by Sie et al. in [4], are: Network Visualization, thanks to *sociograms* which give the possibility to represent networks with nodes connected by edges, directed or undirected; Network Analysis, in which Maths is applied to the network to discover interactions, relationships and so on, using methods at different levels like network, community and individual level; Simulation and Network Intervention, in order to generate a network thanks to some models in which a researcher could be interested in and to modify it to understand the network structure, to create more value from it.

Considering examples of networks mentioned before like *sociocentric* and *egocentric*, there is great interest in the former ones. Mainly researchers are interested in knowing how to discover interactions among users through gathered social data. Looking at figure Fig. 1.2, an example of a users interactions graph generated thanks to Twitter data on a political right-wing demonstration, we should be able to apply analysis at different levels. Considering the most general level we can see that we have a very dense directed graph and a collection of some nodes, probably no more than six or seven, having an high indegree (i.e number of incoming edges in a nodes) mainly positioned in the center of the circle representing the graph. If we pass to the community level, interested in looking for groups on people creating subgraphs and focusing our attention on the top left corner, it is straightforward to discover *cliques* of users (i.e complete subgraph). Cliques in an interactions graph can be interpreted as a group of users who, on a specific topic, intended to communicate with each other instead of joining the whole network interacting with most central nodes or others. Two clear examples of cliques are those connecting users *@siceid* and *@Andrreea* (Twitter ID: 14711801) either with *@seewhatcanbe* (Twitter ID: 16273641) or with *@welikechopin* (Twitter ID: 944251968). If we move to the individual level and analyze specific users, we will discover that some of the most central nodes represent members of the political party that organized the demonstration, user in Fig. 1.3 is a militant of this political party and, after inspecting his profile and some videos on Youtube, we discover he is at house arrest for provoking damages at a comic book fair. Since we are dealing with a political party, it is obvious that the completely political-opposed members would join the conversation. And in fact we discovered the presence of a user connected to a social profile like this in Fig. 1.4.

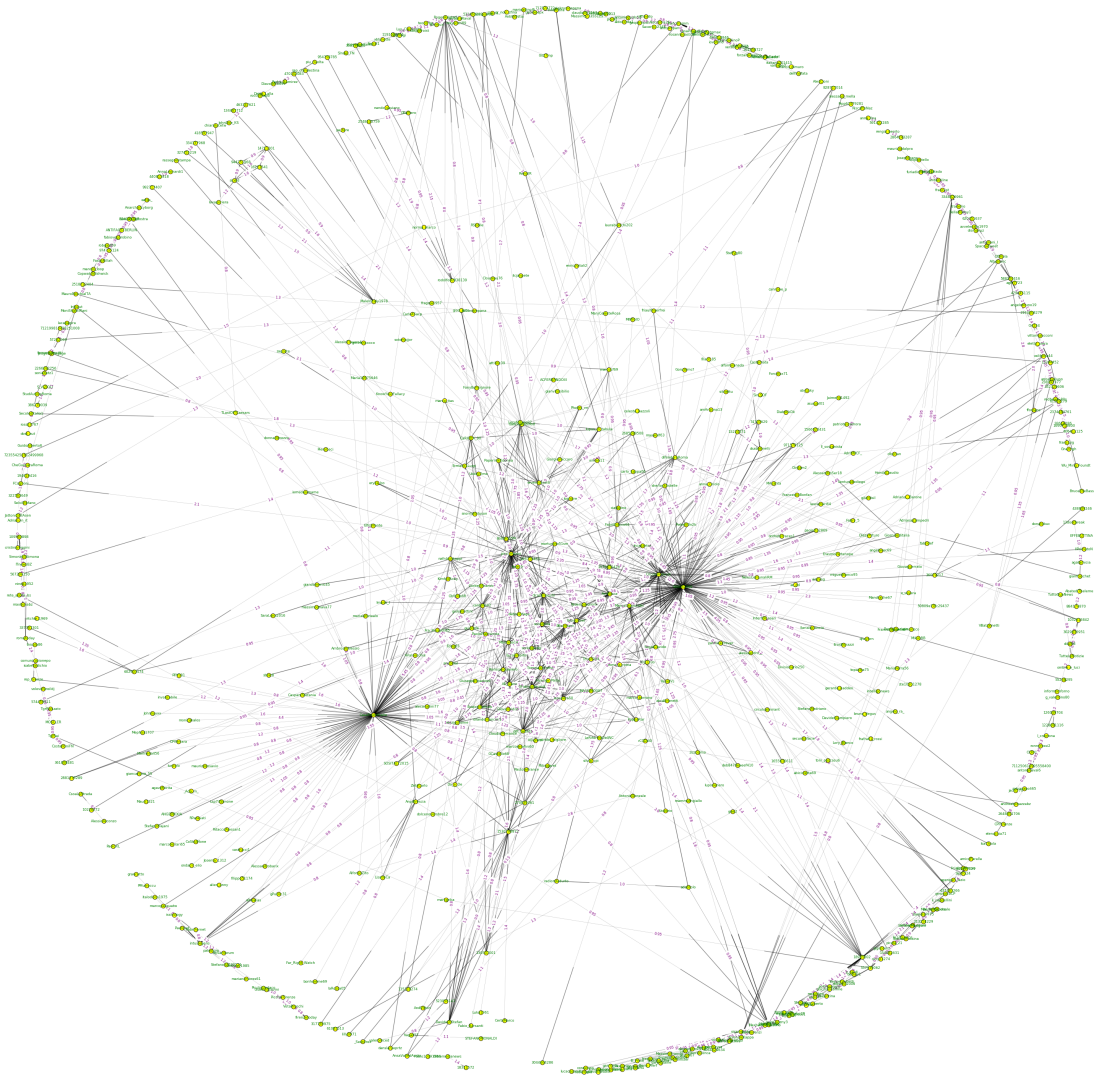

Figure 1.2: *Users Interactions Graph Example*

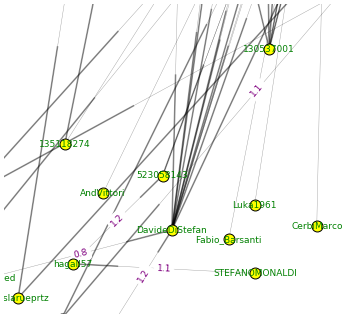

Figure 1.3: *Individual Level Analysis 1*

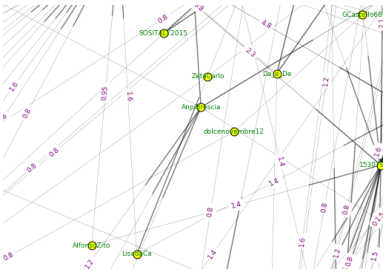

Figure 1.4: *Individual Level Analysis 2*

The example shown presents some characteristics we will discuss later as weight value on all edges. This analysis was simply done by looking at the graph and avoiding calculations. In the case we present a possible application of Social Network Analysis following the guidelines presented in [4]. The interest for this works comes particularly from those enterprises or organizations willing to exploit these mechanism. Possible applications can be found in Intelligence and the previous example is a clear example. Police could inspect this network, look for particular users and monitor profiles considered as subjects at risk. Since there should be a demonstration, it is important to maintain the safety of that part of city and of all people moving around avoiding clashes among people of different factions. Apart this application, we can think of usages in Business Intelligence. An enterprise has just launched a new item and would like to discover if users are happy or not, if they are talking about it, and if users, talking about it, are related to a specific location in the world. Enterprises may discover groups of very happy people enjoying that item or other groups preferring other items of other companies. Social Network instruments may help enterprises in modify their items, proposals, etc.

1.3 Social Media Data Gathering

The starting point for every analysis on social media data is the information retrieval. To address this problem social media usually provide a set of API to third party developers. Retrieved data is affected by the social media and its policies. For example, Facebook, the largest social network in the world, is also one of the most "closed to the world" along with Instagram, from the moment in which it was acquired. Probably one of the most richful of information API is the Twitter's one. It provides information about users, like interests, statuses, followers, lists, likes etc. Since Twitter is one of the most diffused social media in the world, this works focuses primarily on it.

Twitter Data can be retrieved through specific API usage and search. We could be interested in crawling tweets, information about users, locations but also lists of users. In order to crawl data, rules from [5] have been followed. Twitter provides free public API to researchers and practitioners, to retrieve content and users' profile information.

APIs to access Twitter data can be classified into two types based on their design and access method:

- REST API are based on the REST architecture now popularly used for designing web API. Data retrieval has to be pursued with a pull strategy within this API. The request for specific data has to be clearly driven from the user.

- Streaming API provides a continuous stream of public information from Twitter. Data retrieval has to be pursued with a pull strategy within this API. Once a request for information is performed, the Streaming API provides a continuous flow data to be managed by the user.

Twitter API can be accessed only via authenticated requests that are limited to a specific number within a time window. This number is called *rate limit*. After that interval the number of available requests is set again to the maximum possible value in a window interval and requests ca be carried out again. The size of this window is generally 15 minutes.

Twitter uses Open Authentication and each request must be carried out through valid Twitter user credentials. Open Authentication (OAuth) is an open standard for authentication, adopted by Twitter, and other Social Media, to provide access to social information.Since ther is an high risk for passwords to be stolen, OAuth provides a safer method to authenticate using a three-way handshake. The user should be confident in an application using OAuth since his/her password is never shared with third-party applications. The authentication of API requests on Twitter is carried out using OAuth. Twitter API can only be accessed by applications. Below we detail the steps for making an API call from a Twitter application using OAuth:

1. Applications are also known as consumers and all applications require a registration phase Twitter Development Console. Through this process the application receives a consumer key and secret which must be used to authenticate to Twitter.

2. The application uses the consumer key and secret to create a unique Twitter link in which a user authenticates. The user authorizes the application by authenticating himself to Twitter. Twitter verifies the users identity and gets a OAuth verifier, the PIN.

3. The user provides this PIN to the application which uses it to request an Access Token and Access Secret unique to the user.

4. Using both these features, the user is authenticated through the application on Twitter and can start API calls to Twitter.

The Access Token and Access Secret for a user do not change and can be stored in a cache by the application for future requests, avoiding to repeat the same process many times.

1.4 The Big Data issue

Given the astonishing amount of information produced everyday on social media, crawling and processing such data involves specific techniques and technologies. When data reaches a so huge amount of volume, variety and velocity of production, the problem is classified as a big data problem. This term was pretty unknown 15 years ago but probably nowadays is one oh the hottest topic of research.

Kaiser [6] proposed a study in which tried to focus the attention on big data issues and problems. Big data represents those data that cannot be processed efficiently by traditional database methods and tools. The definition goes over the fact that we are now dealing with a huge amount of data, it better seems to go through a definition that creates a new way of looking at data. In next future we won't be interested anymore in data and how it's stored, we'll focus our attention on a more knowledge-centric point of view. This will lead data scientist to work on data avoiding to store it just as they are and trying to enhance the power they can release through the information included in them. So, it will become important to know how to store this information, how to access, manage and process it, possibly in a very fast and efficient way.

The main characteristics of Big Data are:

Data Volume Data volume measures the amount of data available for a requester, that has not be owned for the complete operation time. Since data volume increases, the value of several records may decrease with respect to age, type, importance and quantity, besides other factors.

Data Velocity Data velocity measures the speed in creating, streaming and aggregate those data. Bandwidth may be a problem when dealing with data management and performing extract-transform-load operations.

Data Variety Data variety is a measure of data content like video, images, text, etc. This is probably the greatest obstance since incompatibility in data formats and inconsistency represent significant challenges that can lead to management muddle.

Data Value Data value measures how much data are useful for our own purpose. Data science is focused on the information contained in crawled data as opposed to *analytical science* which undervalues the predictive power of big data.

Complexity Complexity measures the degree of connection and dependency among big data structures themselves. It measures the probability of having a large number of changes due to little modifications in source data. Such changes may affect the behaviour of our current system or may not be significant.

These characteristics lead to:

- *storage* and *transport issues* - the utilized storage may increase very fast in size and the required bandwidth may need to augment if these data increases.
- *management issues* - data have to saved in a storage located in a secure place, where secure means safe in case of disasters, problems and simply accessible by technicians to solve hardware problems.
- *processing issues* - analyzing a huge amount of data means having a great computational power able to execute very fast algorithms on them, possibly on a distribute environment.

Processes which have to deal with big data have to be redesigned in distributed fashion in order to increase performances. We can mention also the fact that, while dealing with Big Data, it is no longer important the quantity of data as opposed to quality of them. Since there are no problems related to data availability, we are more focused on having high quality data that can be used to draw very precise and high-valued conclusions. This is a key point in managing databases along with Social Media and we'll make an important choice in next chapter related to this feature. Another challenge on big data is represented by the compromise between speed and scalability. Executing fast algorithms on a huge amount of data can represent a turning point in this field, but we have to keep in mind that data size we are working on could become huger and huger and probably an algorithm could be affected by this problem. Big data are usually managed by companies such as Social Media who define data structure in their own way. Generally we have to deal with unstructured

data, that may be complete or may miss some values due to inexplicable behaviours of Social Media. Unstructured data gave birth to the so-called NoSQL databases like MongoDB. These database provides a sort of storage in which data is not organized in defined way, data here is stored only to be recovered when the organized data, stored probably in a SQL-like database, has been lost.

2. Related Work

Network among users can be characterized through "static" links which do not evolve (or evolve very slowly) during time, like friendships, or through interactions that continuously occur. The study of the users interaction is the starting point to fully comprehend how users organize themselves in networks on social media. Crawling social media content enables powerful analysis that highlight communities, based on contents and similarity metrics. Contents, which are typically social media specific, allow the creation of an overlay network that describes interactions among users. In this section we want to highlight some previous works which describe how to model interactions among users. The discussion is based on different proposed approaches which inspired our work. Finally, we include other works describing some users characteristics or behaviours, which provide us information for detecting users categories.

A study on interactions among users has been proposed on Facebook by Viswanath et al. [7]. In this work it has been observed that not all links are created equal, while most social networking sites allow only a binary state of friendship. One of the most important study on this field [8] demonstrated that the *strength of links* changes widely, ranging from pairs of users who are best friends to pairs of users who are unfamiliar with each other. In order to distinguish between these strong and weak links, they proposed to examine the activity network, the network that is formed by users who actually interact using one or many of the methods provided by the social networking site. Authors state that it is important to consider time and how interactions vary over time. At the microscopic level, they investigate how pairs of users in a social network interact, and at the macroscopic level, they examine how the single interaction of a collection of those constituting a pattern can affect the general structure of the activity network. This leads to state that users who do not interact frequently are likely to do it thanks to site mechanisms. His group was able to identify mainly these aspects: interactions among users come from pattern of frequent and infrequent interactions. Considering wall posts as interactions among users, it will be an explanation for having too many interactions in user's birthday thanks to Facebook birthday reminder feature. To give

a percentage, it is known that the 54% of interactions are due to birthdays. This kind of interaction is part of the pattern of infrequent interactions. It is simply to understand which interactions are included in the other pattern: mentions, comments and user wall posts. Even when users interact frequently, they find that the activity level of user pairs tends to decrease remarkably over time, implying that most activity links vanish.

Their work is organized this way: first, they only collected information about friendship links starting from a specific user in the New Orleans Network and then they crawled the wall features in Facebook since it represents a broadcast-style messaging service within the site. Their results states there is a strong skew in the total number of wall posts collected between two users; only a little number of users is deeply active on Facebook, instead a great majority interacts only thanks to social media mechanisms or social events. They tried to create two groups of separated users and to take into consideration how much time intervenes between the moment in which they become friends to the first real interaction. They observed that over time interactions between two users shrinks drastically. Conversely, they state that even if individual links and connections lose importance over time, the average network properties remains relatively stable.

Their measurements are clearly not free from limitations. First, in Facebook, users can interact in many ways (e.g., messaging, applications, photo uploads, and chat) and they are not able to say if wall post is a representative method. Second, only users with a visible profile could provide data since Facebook allows users to protect their privacy. Nonetheless, their expectation is that results are representative of all other users in the same network, because their data set covers a majority of the network (66.7%). Third, they exploited the giant connected component of the New Orleans networks in which users are more likely connected; in other networks or communities this work would have been more and more difficult.

Another example working in a similar direction is [9]. Wilson et al. proposed the use of interaction graphs to impart meaning to online social links by quantifying user interactions. They analyze interaction graphs derived from Facebook and show that they exhibit significantly lower levels of the small-world properties shown in their social graph counterparts. This means that these graphs have fewer supernodes with extremely high degree, and overall network diameter increases significantly as a result. They proposed the interaction graph, a model for representing user relationships based on user interactions. They observed that interaction graphs present significantly different properties from those in standard social graphs, including larger network diameters, lower clustering coefficients, and higher assortativity.

This work is important since gives a precise definition of the interaction graph, the milestone of our work. According to them, an interaction graph is parametrized by an two constants N

and T, where N defines a minimum number of interaction events, and T stipulates a window of time during which interactions must have occurred. Taken together, N and T delineate an interaction rate threshold. This consideration let them define an interaction graph as the subset of the social graph where for each link, interactivity between nodes is greater than the rate achieved by N and T. Since a single interaction can be viewed as unidirectional, interaction graphs can contain both directed and undirected edges. It is reasonable for them to represent interactions in an undirected graph since they can show the equality between indegree and outdegree. We won't agree with them and our approach and we'll show a directed version of the *interactions graph*. Moreover, they do not attempt to derive a weight scheme for interaction graphs.

Conover et al. in [10] wanted to investigate how the networked public is shaped in a social media and communities with different political orientations exchange interactions. Working on this they examined data coming from congressional midterm elections. Using a combination of network clustering algorithms and manually-annotated data they demonstrated that the network of political retweets connects the most users coming from the party with the same political vision, having very few connections between the left and the right political side. Conversely, ideologically-opposed individuals interact at a much higher rate through the mention interaction instead of the retweet one. To explain the distinct topologies of the retweet and mention networks they believe that the interaction is provoked by injecting clearly politically defined content whose primary audience is made of politically-opposed users.

Focusing on the two primary modes of public user-user interaction, mentions and retweets as mentioned above, they defined communication links in the following ways. In the retweet network there is an directed edge leaving a node representing user A and reaching a node representing user B if B retweets content originally shared by A, indicating that information moved from A to B. In the mention network an edge goes from A to B if A mentions B in a tweet, indicating that information propagates from A to B (a tweet mentioning B is visible in Bs timeline). Both networks therefore represent potential paths in which information flows between users. We have to underline that in this case they are modelling actions as edges between two users. There is no presence of nodes different from these. In a sense, the interaction graph and the activity graph both resides in the same network as it comprehends retrieved action and involved users. Starting from those tweets they crawled, the first thing they did was to isolate two networks of political communication the retweet network and the mention network. They demonstrated that the retweet network shows a highly modular structure, dividing users into two homogeneous communities corresponding

to the political left and right side. Conversely, they found that the mention network does not present a so high remarkable division between these two political sides, resulting in users being exposed to subjects and information they would not have the possibility to check before being cited. Finally, they provided evidence that these network structures result in part from those politically motivated users who had the sad task of annotating tweets with multiple hashtags and clearly an opinion between two annotators may differ. A good performance can be achieved if some of the experiments are carried out considering inter-rater agreement (IRA)[11], [12] as vital. However this approach leads to an annotation phase in which more tweets are annotated twice or three times and this extra-job and extra-time could not be available.

In [13] Cha et al. presented an in-depth comparison of three measures of influence: indegree, retweets, and mentions. Based on these measures, they investigated the dynamics of user influence across topics and time. They made several interesting observations. First, popular users who have high indegree are not necessarily influential in terms of spawning retweets or mentions. Second, most influential users can maintain significant influence over a variety of topics. Third, influence is not gained accidentally or casually, but through a deep effort such as limiting tweets to a single topic. Their study provides several findings that have direct implications in the design of social media and marketing:

1. Analysis of the three influence measures provides a better understanding of the different roles users play in social media. Indegree represents popularity of a user; retweets represent the content value of users tweets; and mentions represent the name of a cited user. Hence, the top users based on the three measures do not have an high overlap.

2. Their findings on how influence varies across topics could be used as a useful test to know in Twitter how much effective advertisements are and if a user has to be considered valuable in this case or not. The analysis shows that most influential users may have determinant influence over a lot of topics enlarging horizons and avoiding to focus directly on a specific topic, case in which advertisement could really be effective.

3. Ordinary users can gain influence by focusing on a single topic and posting interesting tweets that are nicely considered by others, as opposed to simply conversing, replying and retweeting with others.

Their work clearly do not focus the attention on our goal but it's important because relies on the fact that users are influenced by other users, mainly by those users who have a Verified Account or an high Reputation. This aspect pushes users to specialize themselves on Twitter

when posting or replying. This could help the analysis on an interaction graph thanks to the reliability a user could have achieved through content provided. As said above, if a user gains consideration through specialization on a single topic, could represent a good starting point for our analysis meaning that other users, interested in what he/she posted, would try to interact with him/her and this would generate a possible counter-interaction.

In [14] the work presented by Xiang et al. starts with the consideration that in online social networks the low cost of link formation can lead to networks with heterogeneous relationship strengths (e.g., acquaintances and best friends mixed together) without giving the right value to each interaction. Friendship is considered as a coarse indicator of the interaction between two users and they want to develop an unsupervised model able to estimate relationship strength from activities (e.g., communication, tagging) and user similarity. More specifically, they formulate a link-based variable model, along with an optimization procedure for the inference. Their model relies strongly on maths, since everything is based on precise calculations. Through their model it is shown that the weighted graph formed by the estimated relationship strengths produces an higher auto-correlation and better classification performances than the graphs formed from various aspects retrieved from raw data. This model is so precise that "spurious" link have been considered with a lower weight; conversely the model highlights important links.

A study on the role of interactions on social network in comparison of the friend relationship network is presented in [15], probably the first work in this field. The authors wanted to work of Cyworld, a Korean social network, in order to construct a network from comments written in guestbooks in which a node represents a user and directed edges represents the comment a user produces towards another one. The most important part of their work is the one in which they compare the activity network from a structural point of view with friend network. Topological characteristics of the activity network are similar to the ones of friends network, but thanks to its directed and weighted nature, it allows a more in-depth analysis of user interaction.

Another example of studies related to interactions on social media is the work presented in [16]. This study aims to investigate the relationship between use of Facebook and the formation and maintenance of social capital. The authors worked on a sort of control group, coming from a social environment like an high school and collected information on demographic and other descriptive variables (e.g. gender, age, year in school, local vs. home residence, ethnicity) and Facebook usage measures, such as time spent on Facebook and features to discover if Facebook was an instrument to meet new people or to establish connections with other already connected entities. Facebook usage and indicators of social capital were discovered

highly correlated and in an environment like an high school Facebook, or in general a Social Media, helps maintaining relationships when people move from a community to another.

The big interest manifested towards Social Network Analysis in research field as described above seems pushing people in learning more on what can be created thanks to Social Media Data. Since we want to focus our attention on how users interact on a Social Media, we have to clarify that monitoring only a user or a group of them could be considered apparently a straightforward task. When moving to the analysis of a community of users, Barabasi and his groups in [17] directly states that practically mapping interactions really starts being a complex task. At the beginning this work, but probably also nowadays, was carried out thanks to questionnaire data[18], providing information a sample of users. When Social Media or online community platform started being on the cutting edge, some primal work were instantited on top of these new technologies. It is the case of [19], basing on the role of an activity and on communication networks, authors were able to create and analyze an undirected network completely based on MSN instant messenger services. Those new technologies were able to provide the user a complete infrastructure, mainly in modern Social Media, in charge of creating connections, interact with other members, join communities, participate in discussions. The study of Donath about Social Networks [20] is incredibly important since, not only tries to understand why and how people behave in Social Media or why creates connections or why prefer joining communities, but also lists several design recommendations for networking sites that will be developed in future.

Our work is presented in a period in which many of these considerations have been already applied to social media. This led them to organize themselves in a standardized way. As in Facebook we have posts, in Twitter Statuses, in Youtube videos and so on. In Facebook we have friends, in Twitter Followers, in Linkedin connections and so on. All these considerations pushed us to create a work that can be generalized on top of many Social Media in order to retrieve the interaction graph in which an analyst could be interested in. The ability of recognizing the interaction in which we are interested has to be the key of our work. Exploiting those graphs for Intelligence purposes becomes as essential as due by people working in those fields.

3. Technologies

Dealing with Social Media and data coming from them represents a task that has to be performed through the help of many technologies able to scale when the quantity of data increases. These technologies have to perform many tasks in a very short time because it is undesirable to stretch the queue of data coming from social media. Moreover, they possibly have to write this information in a database maintaining consistency. The list of technologies we are going to present now contains all what has been used to perform our work and to reach the goal of a users interactions graph creation. In the architectural paragraph it will be explained how all these components have been put one after the other in other to understand the whole data processing stage.

3.1 Apache Kafka: Message Broker

Nowadays real-time information is continuously generated by applications (business, social, or any other type), and this information needs straightforward ways to be reliably and quickly distributed to multiple receivers. Most of the time, applications that are producing information and applications that are consuming it are inaccessible to each other and working asynchronously. This leads to redevelopment of information producers or consumers to provide an integration point. Therefore, a mechanism is required to let producers and consumers work themselves and a solution is represented by Apacha Kafka, described in [21] and in [22].

Apache Kafka is a publish-subscribe messaging system implemented as a distributed commit log in a partitioned and replicated fashion, suitable for both offline and online message consumption. It was initially developed at LinkedIn for collecting and delivering high bulks of event and log data with relatively low latency. Message publishing is a mechanism for connecting various applications through messages sending and receiving operations.

Kafka maintains blocks of messages in categories called **topics**. We'll indicate producers as processes that publish messages to a Kafka topic and consumers as processes that subscribe to topics and process the received messages blocks. Kafka is executed as a cluster of one or

more servers each of which is called a **broker**. At a high level, producers send messages over the network to the Kafka cluster which in turn serves them up to consumers like in Fig. 3.1.

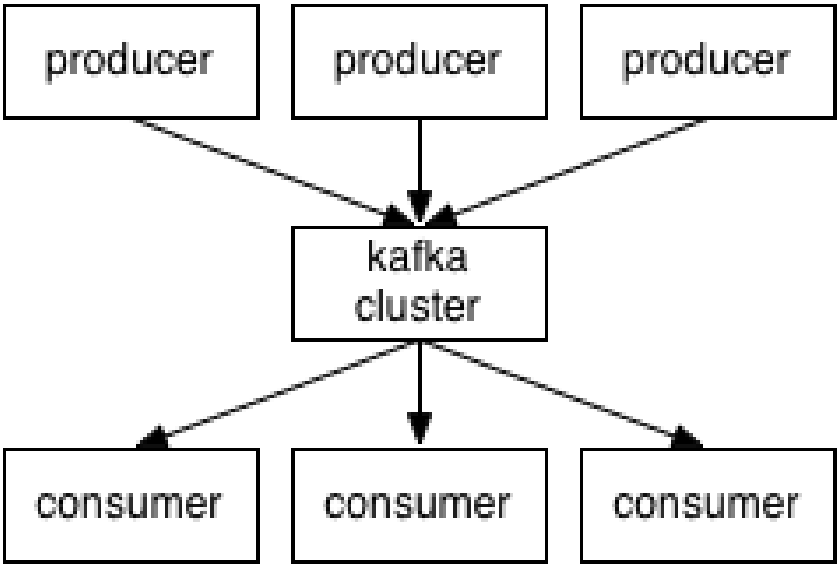

Figure 3.1: *Basic Architecture of Apache Kafka*

For each topic, the Kafka cluster maintains a partition for scaling, parallelism and fault-tolerance. Each partition contains an ordered and unchanging sequence of messages that is continually appended to a log. Messages in the partitions are equipped with a sequential id number called offset. Anatomy of a topic is described in Fig. 3.2

The offset is controlled by the consumer and it will process the next message in the list, although it can consume messages in any order, as Kafka preserves all published messages for a configurable period of time. This makes consumers very cheap, as they can connect and leave the cluster without much impact. Producers are able to choose which topic and which inner partition has to store the produced message. Consumers assign themselves a consumer group name, and each message is delivered to one consumer within each consumer group. Messages are broadcast to each consumer if all the consumers have different consumer groups. Kafka has high throughput, partitioning politics, replication, and fault-tolerance which allows it to deal with Big data. It has the following characteristics:

- High Throughput: To extract the best from big data, information losses are not allowed. Apache Kafka is designed with O(1) disk structures that provide high performance even with very large volumes of stored messages, which could be in order of TB.

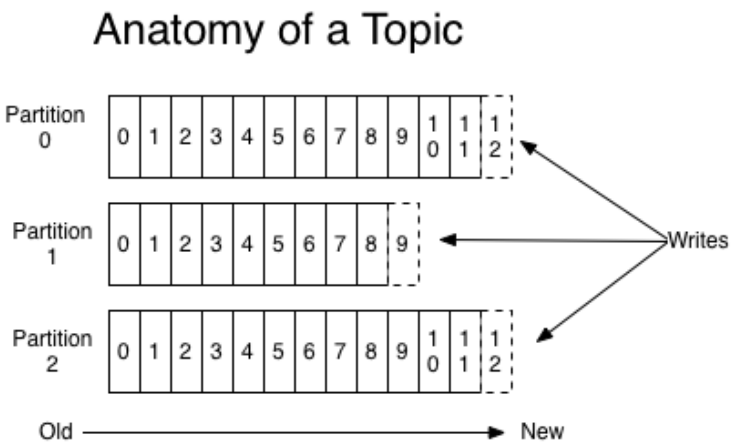

Figure 3.2: *Anatomy of a Kafka topic*

- Persistent Messaging: Considering Big Data, Kafka is designed to work on common hardware infrastructures and to process many messages per second.
- Distributed: Apache Kafka distributes consumption of messages over a cluster of consumer machines while maintaining partition ordering.
- Multiple Client Support: Apache Kafka system supports easy integration of clients developed in Java, .NET, PHP, Ruby, and Python.
- Real Time: Messages produced by threads should be immediately visible to all consumer threads; this feature is very important for event-based systems such as Complex Event Processing (CEP) systems.

Entities dealing with Kafka are Producers, Consumers and Zookeeper.

Producers

Producers publish data in Kafka according to a specific topic and the chosen partition. This can be done in a round-robin procedure simply to balance load or it can be done according to some different algorithm. According to our case, the production of messages follows a random process since they come from social media and it is not possible to predict when a user is producing social content.

Consumers

Messaging traditionally has two models: queuing and publish-subscribe. In a queue, a pool of consumers may read from a server and each message goes to one of them; in publish-subscribe the message is broadcast to all consumers. Kafka offers a single consumer abstraction that generalizes both of these - the consumer group. Consumers label themselves with a consumer group name, and each message published to a topic is delivered to one consumer instance within each subscribing consumer group. Consumer instances can be in separate processes or on separate machines. In our case we configured Kafka as queue since we have only one consumers group having only one consumer in it. This ensures us that only the consumer related to Twitter can download these messages. The system could be configured in the publish-subscribe fashion if more consumers want Twitter messages to create their own self-defined interactions graph.

Zookeeper

ZooKeeper is a centralized service for maintaining configuration information and synchronization in a distributed environment. It has the role of maintaining high-performance coordination service for distributed applications.

To evaluate Kafka and to know if really represents a good solution as message broker, it is clever to exploit statistics Kafka developers provide us in [23]. They conducted an experimental study, comparing the performance of Kafka with Apache ActiveMQ and RabbitMQ, two different message brokers.

Producer Test

A single producer is executed in that test to publish a total of 10 million messages, each of 200 bytes. On average, Kafka can publish messages at the rate of 50,000 and 400,000 messages per second for batch size of 1 and 50, respectively. These numbers are orders of magnitude higher than that of ActiveMQ, and at least 2 times higher than RabbitMQ.

Reasons why Kafka is better are: first, the producer currently sends messages as faster as the broker can handle without waiting for an acknowledgement and increasing the throughput. Second, Kafka has a more efficient storage format. On average, each message had an overhead of 9 bytes in Kafka, versus 144 bytes in ActiveMQ. This means that ActiveMQ was using 70% more space than Kafka.

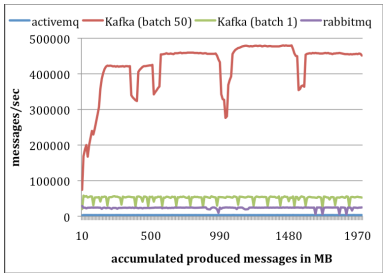

Figure 3.3: *Producer Performances*

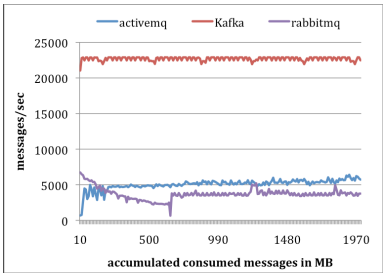

Figure 3.4: *Consumer Performances*

Consumer Test

A single consumer was used to retrieve a total of 10 millions messages. All systems were configured so that each pull request should consume approximately the same amount data. On average, Kafka consumed 22,000 messages per second, more than 4 times better than ActiveMQ and RabbitMQ. This is due to: first, since Kafka has a more efficient storage format, fewer bytes were transferred from the broker to the consumer in Kafka. Second, the broker in both ActiveMQ and RabbitMQ had to maintain the delivery state of every message.

3.2 Titan Db: an example of Graph Database

3.2.1 Graph and Graph Databases

A graph is a structure composed of nodes and edges. Both nodes and edges can have key/value-pairs expressing properties. Nodes denote discrete objects such as a person, a location, or an event. Edges denote connections between nodes. Properties on nodes map non-relational information about the nodes and edges. Example properties include a name or an age of a nodes and an edge having a value. In a graph, each node is considered as an atomic entity (not just a "row in a table") that can be connected to any other node or have new and old properties. This enables the developer to think in terms of users within a world of complex relations as opposed to static tables joined through aggregation operation like in relational databases. Once a domain is modeled, that model must then be exploited in order to extract important information.

A graph database is a software system used to store and process graphs. Nowadays, the common conception in data engineering community is that there is a tradeoff between the scale and complexity of data. Clearly the challenge in this field is to think about a storage able to contain information related to graphs, nodes and edges. Moreover, the developer could be interested not only in creating a graph but also in having nodes and edges of different types,

property not equal for all nodes, etc. The problem is generally solved creating an adaptive framework working on an existing database able to store data and to retrieve it . Without any doubt this could lead to the replication of data in some way, but this is what has been implemented for example in Titan Db, which we are going to present.

3.2.2 Titan Db

General Overview

To challenge this understanding and to generate a database able to work on graph, Aurelius Team has developed Titan[24]. Titan supports both the size of modern data and the modeling power of graphs to perform well in the era of Big Graph Data.

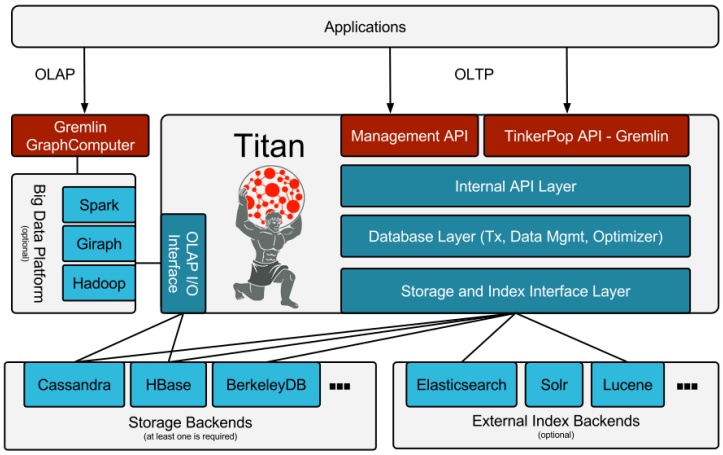

Figure 3.5: *Titan Architecture Overview*

In order to understand how Titan works and its architecture, it is good to say that it relies on many components performing jobs which Titan is only responsible for. On top of Titan and clearly out of it, we have applications running and requesting data to Titan which can be OLAP or OLTP as shown in Fig. 3.5. To access it we can choose to pass through some Big Data Platforms like Spark, Giraph and Hadoop or through API provided to deal with graphs. It's important to say that Titan actual version is 1.0.0 and this makes Titan a young architecture presenting many issues and problems like when facing the interaction with many distributed architectures. For example Titan still does not support a complete way to traverse stored graph through Spark in a distributed fashion and the documentation clearly says that this part is "Coming Soon". In fact nowadays the best way to access it is through the Apache Tinkerpop API which will be described later on. This is how Titan appears to developers

who want to read and write in a graph stored in Titan. Inside it, Titan has some internal API, in order to receive request from outside and to provide an answer, and some connectors able to interface it with 2 important components: the storage backend, which Titan relies on, and an external storage backend. As said above, Titan is not a real graph database, but it can be better thought as a framework or an upper layer which creates a representation of what is stored in the database. To clarify the situation, we have to say that a developer mainly deals with nodes and edges presented by Titan as they have been stored in it. Titan acts as a middleware which creates a graph structure and permits developers to modify and read it.

From developer's point of view, the choice of the storage backend to use is one of the most important configuration when trying to set up an infrastructure storing data in Titan. The choice has to be done depending on which data type the developer is managing and the importance this data has. What he has to consider is definitively the Cap Theorem, which states:

Theorem 1 *It is impossible for a distributed computer system to simultaneously guarantee Consistency, Partitionability and Availability, but maximum two of them.*

- Consistency: requirement specifying that at each moment all nodes on which the database is distributed contain the same data
- Availability: requirement highlighting the characteristic of having a response whenever a request is performed to the database
- Partitionability: requirement enabling the database to work in situations in which a node or more may be out-of-service

Titan was designed in order to have a solution for all possible pairs and, looking at Fig. 3.6, the developer has to choose Cassandra if he does not take care of consistency, HBase if he does not want a storage forever available, BerkeleyDB id he does not have the possibility to rely on all nodes the databse is distributed on. Since our approach requires data from a social media, the very low importance of a single information with respect to the enormous quantity we have at our disposal, the chance of having two data centers on which collect the graph, the strong requirement for the database to be ready to store data at each moment, our choice will be to use Cassandra and discard the other two. A characteristic of Cassandra is to be eventually consistent[25]: it means that in general it is not consistent but at the end it will guarantee this feature. We will describe Cassandra later.

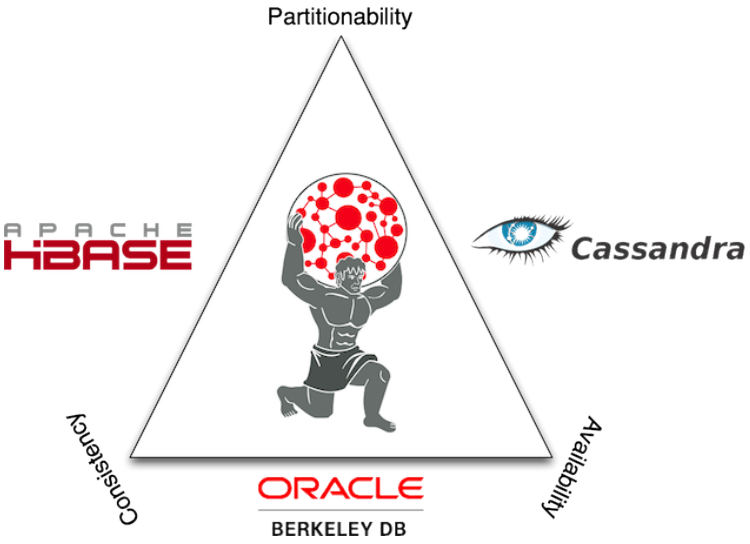

Figure 3.6: *CAP Theorem and Titan*

The choice for the External Index Backend is not related to any particular reason. Titan gives the possibility to choose between Apache Solr and Elasticsearch, which are both based on Apache Lucene. Since we chose Elasticsearch, after there will be a short explanation of it.

Data Structure

Titan stores graphs in adjacency list format which means that a graph is stored as a collection of nodes with their adjacency list. The adjacency list of a node contains all nodes incident edges and properties too. By storing a graph in adjacency list format Titan ensures that data is stored compactly in the storage backend and maintains adjacent lists in a sorted order according to the key to speeds up traversals. The bad particularity is that the information is stored twice, since edges characteristics are stored once per node. Titan stores the adjacency list representation of a graph in any storage backend that supports the BigTable data model as shown in Fig. 3.7.

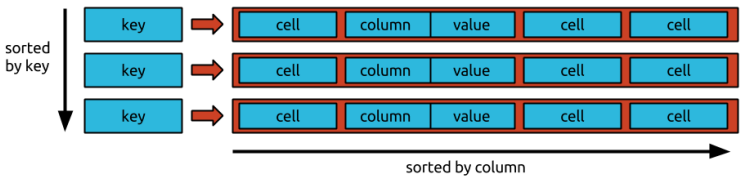

Figure 3.7: *Big Table Data Model*

Under the BigTable data model each table is a collection of rows. Each row is uniquely

identified by a key. Each row comprehends an arbitrary number of cell including a column and value. A cell is uniquely identified by a column within a given row. Rows in the BigTable model are called "wide rows" because they contain a large number of cells and the columns of those cells don't have to be previously defined as it happens in relational databases. Titan has an additional requirement for the BigTable data model: cells must be sorted by their columns and a subset of them, specified by a column range, must be efficiently retrievable. To provide a better loading and traversal performance, Titan can exploit such order to well partition the graph. The storage layout is presented in Fig. 3.8.

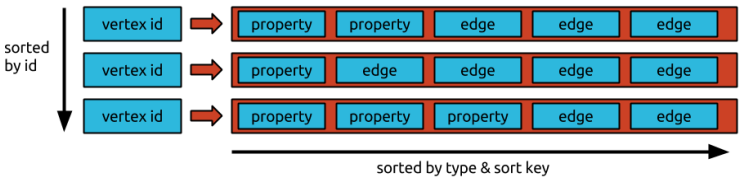

Figure 3.8: *Titan Data Storage Layout*

Titan stores each adjacency list as a row and the 64 bit node id, unique in Titan, is the key which points to the row containing the nodes adjacency list. Each edge and property is stored in an individual cell in the row on which we can perform efficient insertions and deletions. The maximum number of cells allowed per row in a particular storage backend is the maximum degree of a node that Titan can support. If the storage backend supports key-order, the adjacency lists will be ordered by node id, and Titan can assign node ids of an efficient partitioning.

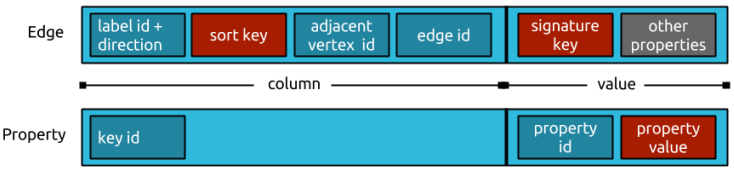

Figure 3.9: *Internal Edge Layout*

Each edge and property are stored in a cell in the rows of its adjacent nodes and the content is serialized. The dark blue boxes represent numbers that are encoded with a variable length encoding scheme to reduce the number of bytes they consume. Red boxes represent one or multiple property values that are serialized with compressed meta data tied to the associated property key. Grey boxes represent property values without compression. The edge has a unique id (as assigned by Titan) represented on a small number. The last bit of this id defines it as an incoming or outgoing edge.

Apache Tinkerpop and Blueprints API

What is important in order to access a graph database is an API that can provide methods to store nodes, to select them, to connect nodes through edges, to look for particular properties in some nodes, to aggregate these results and, in general, to traverse the graph. A property graph is the basic data structure explored and processed by Apache Tinkerpop. TinkerPop is a graph computing framework written in Java. TinkerPop provides a core API that graph system vendors can implement. There are various types of graph systems including in-memory graph libraries, OLTP graph databases, and OLAP graph processors. For many, TinkerPop is seen as the JDBC - Java DataBase Connectivity - of the graph computing community. To ensure ease of adoption by this community, Titan natively contains some components dedicated to many tasks as said in [26] and shown in Fig. 3.10:

- Rexster: A graph server to make a graph accessible via REST or a Rexster-specific binary protocol
- Furnace: A graph algorithms package exposing property graphs to single-relational graph algorithms
- Frames: A component exposing the elements of a graph as Java objects. Instead of writing software in terms of nodes and edges, it can be written in terms of domain objects and their relationships to each other
- Gremlin: A domain specific language for traversing property graphs. Gremlin makes use of Pipes to perform complex graph traversals.
- Pipes: A dataflow framework that enables the splitting, merging, filtering, and transformation of data from input to output.

All these components have been merged when Tinkerpop began to work on Blueprints, a generic Java API for graph databases, passing to Tinkerpop 3. Blueprints is a property-graph model interface providing implementations, tests and supporting extensions. Blueprints defines a particular graph model, the property graph, which is basically a directed and labeled multi-graph:

$$G = (V, I, \omega, D, J, \gamma, P, Q, R, S)$$

where V is a set of nodes, I a set of nodes identifiers, $\omega : V \rightarrow I$ a function that associates each node to its identifier, D a set of directed edges, J a set of edges identifiers, $\gamma : D \rightarrow J$

a function that associates each edge to its identifier, *P (resp. R)* is the nodes (resp. edges) attributes domain and *Q (resp. S)* the domain for allowed nodes (resp. edges) attributes values.

Generic operations provided by Blueprints are: add and remove nodes, retrieve node by identifier (ID), retrieve nodes by attribute value, etc.

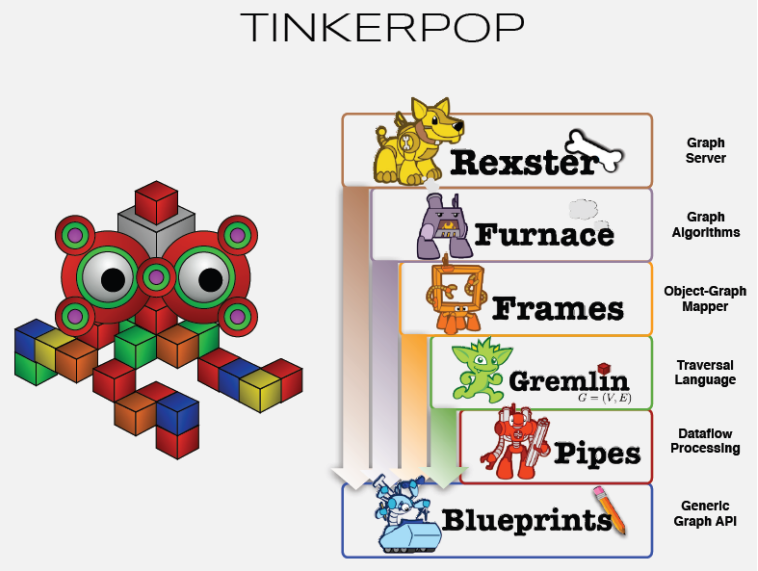

Figure 3.10: *Tinkerpop Components*

Working only with Blueprints in an application allows to use a graph database being unaware of the implementation, making it possible to easily switch from a graph database to another one without too much effort. Drawbacks are related to the automatic performing of some actions. For instance, it is not possible to specify the starting point of a transaction: it is automatically started when the first operation is performed on the graph.

Gremlin Traversal Language

The component that survived to the merging operation which gave birth to Blueprints and still now is main part in Tinkerpop is the Gremlin Language which is described in [27]. Gremlin can be considered as a graph traversal traversal machine and language. The main characteristic of a query language for a graph database is that it has to embed either a graph traversal or a graph pattern match perspective. In the traversal approach, the query returns as result a list of entities, or better, locations like nodes, following particular user provided instructions where traversers halted their path. In the other approach the user defines some variables in a subgraph and all graph elements which are linked to those variables constitute

the result. Gremlin supports both approaches and its machine and language structures naturally facilitate Gremlin being

1. embedded in a host programming language
2. extended by users willing to simplify their inspection
3. optimized thanks to inner rules
4. executed within a multi-machine cluster
5. valuated in different orderings like breadth or depth-first
6. represented within the graph itself via the theoretical existence of a Universal Gremlin Machine.

Gremlin, as a graph traversal machine, is composed of three components: a graph G (data), a traversal Ψ (instructions), and a set of traversers T (read/write executors). Conceptually, a collection of traversers in T move in the core of G according to the instructions specified in Ψ. Since we have a multi-relational graph characterized by attributes too G=(V,E,λ), where V represents veritces, E represents a set of edges

$$E \subseteq (V \times V)$$

$$\lambda : ((V \bigcup E) \times \Sigma^*) \rightarrow (U \setminus (V \bigcup E))$$

λ represents a partial function that mas an element/string pair to an object in the universal set U. Given λ, every node and edge can have an arbitrary number of key/value pairs called *properties*.

A **traversal** Ψ is a tree of functions called steps. The steps organization can be done these ways:

- **Linear motif**: The traversal $f \circ g \circ h$ is a linear sequence of three steps where the output traversers of f are the input ones of g. The same happens for g and h.
- **Nested motif**: The traversal $f(g \circ h) \circ k$ contains the nested traversal $g \circ h$ which is an argument of the step f. In this way, f will exploit $g \circ h$ in its mapping of its input traversers to its output ones which are then presented as input to k.

A step $f \in \Psi$ defined as $f : A^\star \to B^\star$ maps a set of traversers identified by objects of type A to a set of traversers identified by objects of type B. $A^\star$ notation specifies that multiple traversers may be the same element contained in A. The Gremlin graph traversal language defines approximately 30 step functions, but all of them can be referred to the 5 general types as stated below:

- **map** : $A^\star \to B^\star$, where $|map(T)| = |T|$. These functions map the traversers T at objects of type A to a set of traversers at objects of type B without altering the traverser set size.
- **flatMap** : $A^\star \to B^\star$, where the output traverser set may be smaller, equal to, or larger than the input traverser set.
- **filter** : $A^\star \to A^\star$, where $filter(T) \subseteq T$. The traversers in the input set are either retained or removed from the output set.
- sideEffect : $A^\star \to_x A^\star$, where $sideEffect(T) = T$. An identity function operates on the traversers though some data structure x is mutated in some way.
- branch : $A^\star \to^b B^\star$, where an internal branch function $b : T \to P(\Psi)$ maps a traverser to any number of the nested traversals start steps.

A **traverser** unifies the graph and the traversal through a reference to an object in the graph and a reference to a step in the traversal. A traverser can be expressed as:

$$T \subseteq (U \times \Psi \times (P(\Sigma^\star) \times U)^\star \times \mathbb{N}^+ \times U \times \mathbb{N}^+)$$

The expression above seems very complicated but it can be explained this way: The first element represents the location in the graph for the traverser; the second element is the location in which will be the traverser after a traversal step; the third element comprehends a sequence of strings and objects called *labeled path* specifying all traverser's steps in the path labelled by a word; the fourth element stands for the traverser *bulk*, the number of traversers represented by our traverser; the fifth element is a local variable of the traverser called traverser's *sack*; the sixth element counts the number of times a traverser has encountered a *loop sequence*, variable called *loop sequence*. To explain better we can specify one by one all those functions as:

1. μ: $T \to U$ maps a traverser to a location U
2. ψ: $T \to \Psi$ maps a traverser to a step in Ψ ,the location in the traversal

3. Δ: $T \to (\mathcal{P}(\Sigma^\star) \times U)^\star$ maps a traverser to its labeled path

4. β: $T \to \mathbb{N}^+$ maps a traverser to its bulk

5. ς: $T \to U$ maps a traverser to its sack value

6. ν: $T \to \mathbb{N}^+$ maps a traverser to its loop counter

To summarize, a traverser can be seen as a collection of local variables mapping itself into a location in the Graph and a location in the traversal.

$$G \leftarrow \mu \ \frac{t \in T}{\{\Delta,\beta,\varsigma,\nu\}} \ \psi \to \Psi$$

Gremlin admits a user to define a traversal Ψ and to start traversing the tree using functions called in with very clear name in order to move in the graph. Gramlin has the possibility of three different variants: Gremlin-Groovy [28], Gremlin-Java [29], Gremlin Scala [30].

We can now present a little example in which an traversal is defined and we take a look at results and at the steps which are performed to complete it. Let us consider a case in which we have only to traverse the graph an reach a node or a list of them satisfying our request. We are interested in knowing the subject known by a certain user, call him Marco, which is older than the others known by him.

```
g.V().has("name","Marko").out("knows").values("age").max()
```

We have now to process the query executing all steps one by one. We have to start from the first object which is g representing the traversal. First operation *V()* is the definition of the traverser set bijective to V, where $\biguplus_i \mu((\mathrm{V}_g)_i) = \mathrm{V}$. If we would be interested in having a sort of SQL-like notation, the sintax would be:

$$\max(\mathrm{values}_{\mathrm{age}}(\mathrm{out}_{\mathrm{knows}}(\mathrm{has}_{\mathrm{name=marko}}(\mathrm{V}_g))))$$

Once we've got all nodes, we are interested in having a list of those nodes having name equal to Marko. At this moment we could get as result a list of traversers or just one, referencing those nodes named this way. Next step can be processed exploring all edges starting from the node we have as sources and looking for reached nodes. Here we are processing a one-step operation but generally it can be repeated many times jumping from the beginning node to the node which we are interested in. We are moving from a parent node to a children node located at those nodes that are outgoing *knows*-adjacent to the *marko*-node. The children have a new graph location, traversal step location, and a path that is the concatenation of their parents path and their current location. A tuple representing this operation would be:

$$\texttt{(y, values}_{\texttt{age}}\texttt{, ((}\emptyset\texttt{,x),(}\emptyset\texttt{,y)), 1, }\emptyset\texttt{, 0)}$$

where λ(x,name) = marko and λ((x,y),label) = knows. The last operation maps to a traverser set where each child traverser is located at the integer value of their current nodes *age*-property. Finally, max() transforms the traversers at $\mathrm{N}^\star$ to a single traverser at a number representing the maximum number in the previous set.

$$\begin{aligned} V_g &: 0 \to \mathrm{V}^\star && \text{flatMap} \\ \mathrm{has}_{\mathrm{name=marko}} &: \mathrm{V}^\star \to \mathrm{V}^\star && \text{filter} \\ \mathrm{out}_{\mathrm{knows}} &: \mathrm{V}^\star \to \mathrm{V}^\star && \text{map} \\ \mathrm{values}_{\mathrm{age}} &: \mathrm{V}^\star \to \mathbb{N}^\star && \text{map} \\ V_g &: [\mathbb{N}^\star] \to \mathbb{N} && \text{map} \end{aligned}$$

Statistics and Comparisons with other Graph Databases

The decision to choose Titan as database for this work was not simple and mainly based on [31]. In that work is presented a test they developed on Titan and other databases like Neo4j, DEX and OrientDB. The first reason which lead us to choose Titan is that it is based on many Apache components as we saw above. We can rely on Tinkerpop, Blueprints and Gremlin for the part related to the API, Cassandra as storage backend, Elasticsearch as external index backend. The second reason is due to the fact that Neo4j, DEX and OrientDB are not open-source, a particular characteristic that instead Titan has. Then we evaluated the mentioned work and, apart all technical characteristics their infrastructure has, we discovered they evaluated basically three type of operations:

- Load workload: it starts a graph database and load it with a particular dataset. Graph elements (nodes and edges) are inserted progressively and the loading time is measured every 10,000 graph elements loaded in the database. Moreover, the user can control the loading buffer size, that simply indicates the number of graph elements kept in memory before flushing data to disk.

- Traversal workload: perform a particular "traversal" on the graph. They concentrated their traversal on a shortest path workload (find all the paths that include less than a certain number of hops between two randomly chosen nodes) and neighborhood exploration workload (find all the nodes that are a certain number of hops away from a randomly chosen node)

- Intensive workload: a certain number of parallel clients are synchronized to send together a specific number of basic requests concurrently to the graph database. Two of this queries regards the "GET" operation on nodes or edges by ID or property, the third regards a "GET" operation and an "UPDATE", the last regards a pair of "GET" operations on nodes and an "ADD" operation on a edge between these two nodes.

Titan was evaluated either with Cassandra or with BerkeleyDB as storage backend. It resulted that:

- In the "Load Workload" test, all databases experienced short time to complete the whole loading phase, except for OrientDB which, after having inserted 500000 nodes, requires a time increasing exponentially.
- The "Traversal Workload" is the most scary one because states that Titan-Cassandra is probably the worst solution if you have to deal with an intensive traversing phase. Other databases did not show so many problems till when the number of hops was going over 4
- The "Intensive Workload" was positive since it resulted that, particularly for the operation of "GET & ADD" Titan-Cassandra resulted as the best, showing a great performance with respect to mainly Neo4j.

Considering that our work won't require a deep exploration of the network, with a maximum number of hops equal to three, the choice of Cassandra is the right one. This is a good choice also because we'll have a great number of "ADD" operation and Cassandra really performs well in this case. If it seems not clear, we'll advice to look at paragraph on architecture to understand better our work.

3.2.3 Cassandra

Apache Cassandra[32] is a distributed storage system for managing very large amounts of structured data distributed across many nodes spread on several data centers, while providing highly available service with no single point of failure. The way Cassandra manages the persistent state with respect to failures pushes many platform to choose it as storage backend. While in many ways Cassandra can be considered as database and shares many design and database implementation strategies, Cassandra does not support a complete relational data model; instead, a simple data model is presented to developers in order to have a dynamic control. Cassandra system was designed to be executed on not so expensive hardware, in

fact it was used by us on two stand-alone desktop virtual machines, and handle high write throughput while not reducing read efficiency. It is a NoSQL database that was initially developed by Facebook and powered their Inbox Search.

Cassandra provides a storage based on key-value records with tunable consistency. Keys map more than one value, which are grouped into column families. These column families are generated when the database is created in Cassandra but we can add and remove them whenever we want. The values from a column family for each key are stored together and this let Cassandra be a hybrid data management system organized between a column-oriented fashion and a row-oriented one.

One of the key design features for Cassandra is the ability to scale when data quantity increases. Cassandra distributes data on the cluster using consistent hashing algorithms however maintaining an order. The architecture of Cassandra is based on a ring infrastructure. Each node on the system has a random value which represents its position in the ring. Each data item identified by a key is sent to a node by hashing the data item's key to retrieve its position on the ring, and then walking the ring to find the first node with a position larger than the item's one. This node becomes the coordinator for this key. The most important challenge of this hashing algorithm is that we have a non-uniform data and load distribution thanks to the fact that each node has a random position in the ring.

To achieve high availability and durability, Cassandra uses replication over nodes. Each data item is replicated at N nodes, where N is a replication factor configured each time in a different way. Each key, k, is assigned to a coordinator node. The coordinator has the due to replicate data chunks that are stored in its range. Cassandra guarantees durability when a node or network partitions fails by relaxing previously defined requirements. Data center failures may be caused by power outages, cooling failures, network failures, and natural disasters. Cassandra has the characteristic to be configured such that each row is replicated across multiple data centers. However a Failure Detection Algorithm has been implemented in order to detect and solve failures. It is a mechanism by which a node can determine on its own if any other node in the ring is up or down. Cassandra uses a modified version of the ϕ Accrual Failure Detector[33]. The idea of an Accrual Failure Detection is that the failure detection module doesn't emit a Boolean value establishing whether a node is up or down. Instead the failure detection module emits a probabilistic value which represents a suspicion level for each of monitored nodes.

The Cassandra system guarantees data persistence relying on the local file system. The data is represented on disk thanks to a format able to good performances in retrieval operations. Clearly, being a database, it has to deal with writing and reading operations. When a

read/write request arrives at any node in the cluster Cassandra identify nodes, route the requests to the nodes and wait for a response and if it does not arrive in a short time window, return a error response to the user, trying to recover the problem.

Cassandra main characteristics are: **Decentralized** (Every node in the cluster has the same role. There is no single point of failure. Data is distributed across the cluster such that each node contains different data), **Supports replication and multi data center replication** (Replication strategies are configurable. Cassandra is designed as a distributed system to let a user work on multiple data and to able to recover from disasters), **Scalability** (Read and write throughput both increase linearly as new machines are added, with no downtime or interruption to applications), **Fault-tolerant** (Data is automatically replicated to multiple nodes for fault-tolerance. Replication across multiple data centers is supported. Failed nodes can be replaced with no downtime).

3.2.4 Elasticsearch

Elasticsearch, described in [34], is an open source full-text search engine written in Java. Its main characteristics are related to the distributive, scalable fashion in which it has been implemented, but also to the fact that it is capable of approximately real-time search. A running instance of the Elasticsearch server is called a node and, when connected to two or more nodes, it can create an Elasticsearch cluster. Once configured, Elasticsearch is able to discover on its own cluster nodes and to create connections among nodes.

Indices in Elasticsearch are the elements storing all data. An index can be compared to a database in a RDBMS: it can store documents of different types, update them, and search for them. A document consists of zero or more fields, where each field is either a primitive type or a more complex structure. A Document type is assigned to each document; however, all documents in Elasticsearch do not have a precise schema, which means that two documents of the same type can have different fields. Document type can be compared to a table in RDBMS: it defines the fields that can be specified for each document.

Elasticsearch element	SQL element
Index	Database
Mapping	Schema
Document type	Table
Document	Row

Table 3.1: *Elasticsearch vs. SQL: Comparison*

Elasticsearch is based on Apache Lucene; each Elasticsearch index consists of one or more Lucene indices, called *shards*. Before creating the index, we have to set the number of *shards* that each index has. When a document is added to an index, the Elasticsearch server assigns this document to a shard. This shard will have to store and index this document. Elasticsearch balances the loads between available shards but also distributes all shards among the nodes in a cluster, having a perfect balancing in shards and nodes.

Elasticsearch is a RESTful server, so the main way of communication with it is through its REST API. Communication between server and a client is straightforward since there are no restrictions on programing language used to implement those clients or the platforms that they operate on. One of the main element in Elasticsearch is a *mapping*, which is similar to a schema definition in SQL databases. The crucial characteristic of a *mapping* is that it defines all document types assigned to an index and how each document and its fields are stored, analyzed, and indexed.

Points of strength of Elasticsearch are represented by: **Scalability** (Elasticsearch automatically distributes shards of an index over the nodes of a cluster and controls that the loading phase is performed equally over them), **Agility** (New records and updates are performed rapidly as well as changing structure of a logical piece of a document. Moreover, Elasticsearch does not impose schema on the documents in indices. If a new document is added to an index with a different fields schema, Elasticsearch will automatically update the mapping), **Performance** (Queries in Elasticsearch are performed real-time thanks to the fact that indices are refreshed in short time intervals). Weaknesses of Elasticsearch are concerning **Security** (Elasticsearch lacks security features such as authentication and access control. If an adversary knows the URL of the server, he can easily delete all the indices and shut down the cluster).

To recap, it is clear that Elasticsearch is a good choice if we expect to perform any real-time analysis of the data such as as social media analysis. In fact Facebook and Netflix adopted it as index storage backend.

3.3 Brief Scala Overview

The language used to develop parts of this work is Scala, created in 2011, very well described in [35] and overseen in [36]. Scala fuses object-oriented and functional programming in a statically typed programming language. It is aimed at the construction of components and component systems. Scala is an acronym for Scalable Language. This means that Scala grows with the increasing number of data to process. It is perfect when dealing with enormous

quantity of data coming, for example, from social media. The most important feature of Scala is that it is 100% compatible with Java and it runs on the JVM too: this means that it is possible to write a program in Java and it can be converted in few steps in Scala [1]. Java and Scala classes can be freely mixed, no matter whether they reside in different projects or in the same. They can even mutually refer to each other, the Scala compiler contains a subset of a Java compiler to make sense of such recursive dependencies. Moreover, all libraries, frameworks and tools a developer has to use in Java can be simply imported in Scala too and all data types and Java data structure can be used also in Scala. The contrary is not true but it is wise to think that Scala, in a sort of way, embeds Java. As far as Scala is used as a scalable language and it is used to deal with data, it is stateless and tries to minimize, with the respect to other coding languages, the amount of memory used in programs. This can be clear seen when using data structure like array, lists or other objects, defined "immutable", to store a lot of simple objects. Scala does not permit to assign a value to a variable stored in array and changing it after. This is shown also in the declaration of a simple object, because in Scala everything is an object, functions too. Scala advices to define it as a "val" object that means to assign this object a values that will remain the same for the whole program, as if we define it with a final in Java or const in C. Clearly it is possible to declare an object that has to change it's value during the execution of the program and it can be done through the "var" statement, even if this operation is deprecated in Scala. The language is designed to be clear and concise, with various implicit techniques to help simplify common tasks. Scala is particularly brilliant when it is used in scalable server software that makes use of concurrent and synchronous processing, parallel utilization of multiple cores, and distributed processing in the cloud. Performant multi-threaded code can be developed through its functional nature. Apart from having all this benefits, Scala has also some drawbacks like being a little bit criptic when programming through IDE like IntelliJ Idea. Sometimes understanding methods and parameters becomes really complicated. In Scala, differently from Java, the same thing can be done in many ways. This is clear when using complex data structure and ways to access data contained in it. When programming it can lead to confusion.

[1] http://javatoscala.com/

4. Model

The work, we would like to perform, consists of some fundamental steps we have to carry out in order to produce the users interaction graph. It starts with the creation of an *activity network*, that maps all activities users are allowed to perform on a social media, and a *interaction graph* which derives from the previous network and, in a sense, translates the information to show in which way and how much users interacted in a social media. This steps requires the definition of a weight schema in charge of mapping each activity to a specific interaction value. At the beginning we have to clarify what is the interaction for our work. This model would propose a general system that, given a particular definition for the interaction, should be able to provide the *activity network* and its consequent *interaction graph*. The possibility to define the interaction in a complete different way, depending on what a researcher is interested in, enables this system to be general-purpose and able to be adapted to different use cases. A general definition of interaction could be:

Definition 1 *An **Interaction** is a connection between two specific users. It can be direct or indirect and is characterized by a value, representing its strength.*

4.1 Activity Network

As said before, the very first thing to do is to define the interaction. In this model our definition of interaction derives from the production of a tweet a user produces in Twitter platform. All the activities that are generated by the production of this content have to be mapped in the *activity network*.

Definition 2 *An **Activity Network** is a directed graph containing nodes, representing the entities provided by the social media, and edges connecting those nodes, expressing the activity performed by a node N_i towards N_j.*

Before talking about this, it is advisable to list all activities that involve users in Twitter. Many of the cited subjects below have an explicit meaning, others like lists represent collec-

tions of users called members. Subscribers of a list are those users who want to be notified on content shared by members. Activities are listed below:

- User A follows User B
- User A is followed by User B
- User A posts a content specifying an Hashtag H
- User A posts a content mentioning User C
- User A replies to User B
- User A retweets to User B
- User A likes a post of User B
- User A mutes User B
- User A blocks User B
- User A reports User B
- User A can be a member of a List L
- User A can be subscribed to a List L

This is brief list of what is possible to do on Twitter. Clearly we might not be interested in all these activities. The first reason is that some of them have such a weak relevance that in our study can be neglected; the second reason is that some activities cannot be crawled because there are no REST API for them like mute/block/report; the third reason is that some interactions like follow/followed are the essence of a complete different approach and, since we are interested in activities not in pure friendship relationships, we won't model them. Moreover, this activities do not require a tweet. After these considerations, we have to say that everything is contained in a tweet can be considered as something useful to store in the database and to create the *activity network*.

We have now to define the type of nodes and edges connecting those nodes according to our system:

- *User U*: representing a Twitter User
- *Tweet T*: representing a Twitter Status

- *Entity E*: representing an Entity that can be embedded in a status like a video, an image, an url or an hashtag

Entities are key-nodes and are modeled in the network as shown in Fig. 4.1, since they can provide information or may allow some special searches. The figure shows that everything in the model stands around the tweet. It contains zero or more entities, it could mention zero or more users, it is produced by a user, it can optionally retweet or reply to another tweet.

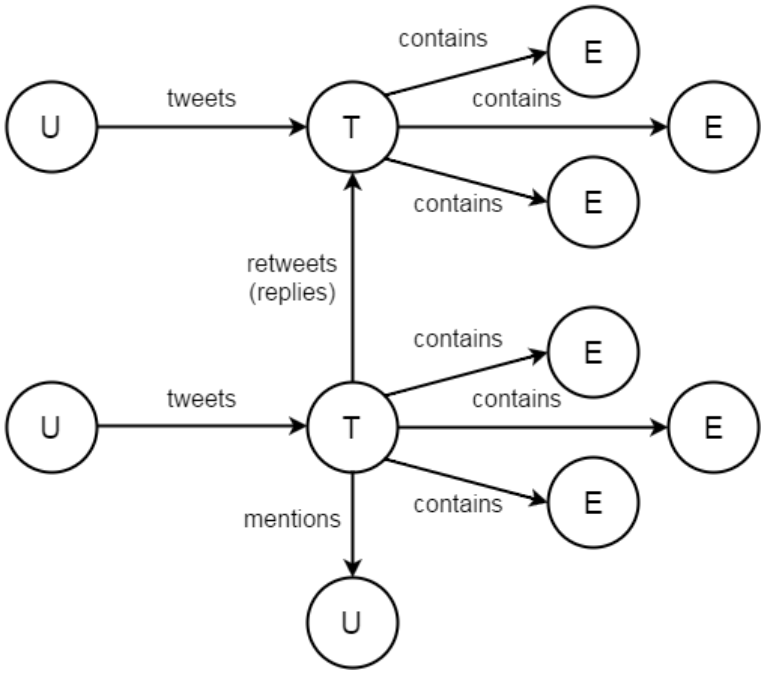

Figure 4.1: *Tweet Activity Network Example*

Edges connecting nodes map those activities that can be performed around a content and involve nodes. Considering a theoretical point of view, activities may be compared to associations of a E-R Diagram. Each association may have a cardinality, called *multiplicity* in graphs, expressing how many times occurrences of source node may be tied to occurrences of destination node. In a simpler way, it highlights if a node may have one or more incoming or outgoing edges from a connected node. Activities can be listed like:

- *tweets*: action accomplished by a user and has a ONE TO MANY multiplicity
- *contains*: link that goes from a tweet to an entity and has MANY TO MANY multiplicity
- *mentions*: link specifying that a user is mentioned by a tweet and has MANY TO MANY multiplicity

- *retweets/replies*: action that connects a tweet to another one and has a multiplicity MANY TO ONE

At this moment, we have to define a constraints schema in order to maintain the consistency with the source social media and in order to have data really representing what is present in it. This schema is related to the considered Social Media and it has to be defined by scratch each time. Examples of this schema can specify the number of users that can produce content, the number of entities that may be contained in a content or if a content may share more than one content. There is a list of characteristics in this model that have to be mentioned in order to create correctly the graph and respecting Twitter liens:

- The author of a Tweet can be only one specific User, since more producers for a single a tweet are not allowed, and it contains a unique tweet id.
- A tweet can reply or retweet only one tweet.
- A tweet can contain zero or more entities, can mention zero or more users.
- Tweets specifying a mention at the beginning of the text field do not have been modelled as replies, as Twitter does, but as simple mentions.
- An entity can be contained in zero or more tweets.
- There no direct relationships among users.

An example of an Activity Network is shown in Fig. 4.2, where it is possible to see how users, tweets and entities are connected through Twitter allowed functionalities.

4.2 Users Interactions Graph

The Activity Network can be considered a block that substitutes all data coming from the Social Media and gives us the perception of what we got as data. The next step regards the creation of a users interactions graph letting us be aware of how users interacted in this data set based on our search.

Definition 3 *A **Users Interactions Graph** is a graph containing edges which connect users, denoted as nodes, and expressing a value on each edge that represents the interaction between two specific users.*

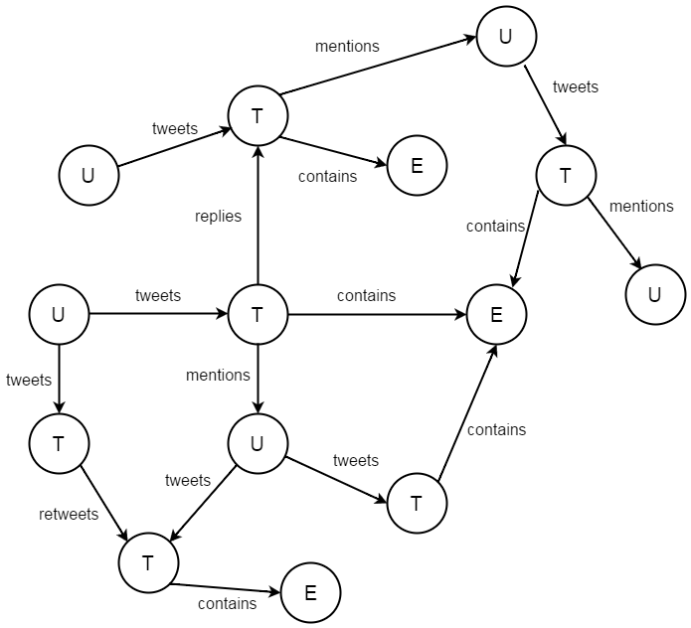

Figure 4.2: *Activity Network Example*

After a deep analysis on the allowed activities on Twitter and a reasoning on the social value of the interactions to map, our approach considers two types of interactions: direct and indirect. Their definition is:

Definition 4 *A **direct interaction** is the one that can be established when, given a user U_j and U_i , U_j performs an activity towards U_i.*

Definition 5 *A **indirect interaction** is the one that can be established when, given at least three users U_j, U_i and U_y, U_i and U_y both perform an activity towards U_j or both undergo it from U_j.*

To give examples of direct interactions, we have to say that, when a user U_i mentions a user U_j, replies or retweets a tweet of user U_j, a direct interaction has been generated. For indirect interactions we intend those interactions that are not involving two users in a clear way but those interactions are hidden in tweets. As indirect retweet and reply interactions we want to specify those situations in which two users retweeted or replied the same tweet and, in a sense, they should result as involved in the same conversation. As indirect mention instead, we want to model those situation in which more than one user is mentioned in a tweet. Why should these interactions be important and why did we decide to model them? The reason is that if we consider them as stand-alone interactions, their importance is really low, but consider the moment in which two users, mentioned in the same post, start replying each other

on the same tweet or on others too. Clearly we'll see the values of both edges augmenting but it could be interesting to understand where, when and why this new interaction started. Firstly, this could give birth to a new interaction; secondly, this could present to the network analyzer the reason of both mentions and the tweet content that involved both users.

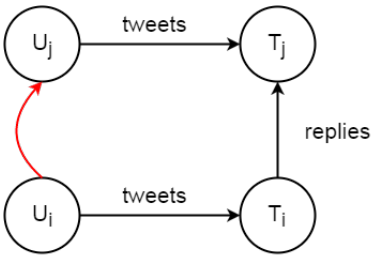

Figure 4.3: *Direct Interaction Example 1*

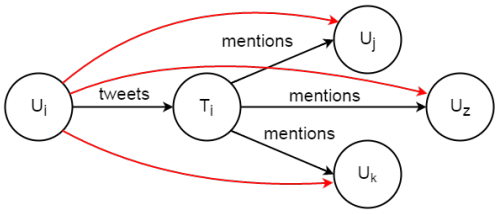

Figure 4.4: *Direct Interaction Example 2*

The graph can be directed, specifying that a user has an interaction towards another, or undirected, simply modelling the interaction between two users. Clearly the chosen way to represent it creates a different intrisic meaning. If we choose the undirected representation, the value on the edge will comprehend all interactions found in the previous graph without clarifying if the number of those interactions going from user U_i to user U_j is greater, lower or equal to the number of those going from U_j to U_i. Conversely using a directed graph, each edge represents the sum of all weights related to those interactions going from user U_i to user U_j. Another edge, going from user U_j to U_i, embeds all contributes coming from those interactions in this direction.

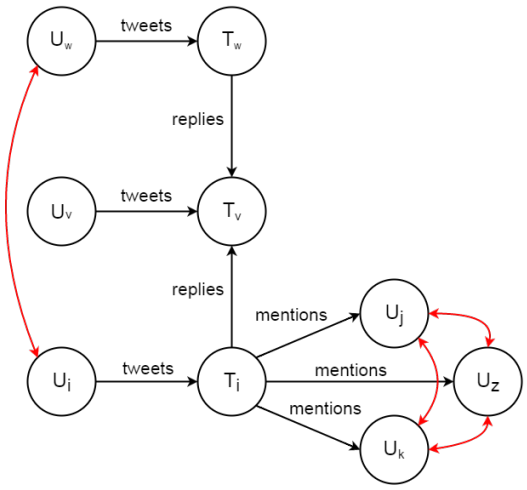

Figure 4.5: *Indirect Interaction Example*

Modelling the graph in a directed way surely represents a proper solution. Two main

reasons got us to this choice. Firstly, modelling the graph in an undirected fashion could lead to situations in which the value on the edge could be really high, letting us think that those two users are really tied each other, but probably that number could be so high thanks to interactions going only from a user to the other: this is the case of Verified Accounts in social media, which produce a so high number of contents and get an increasing number of shares and comments from a great numbers of users interested in that topic, without answering directly to any of them. This clearly will create a high value on all those edges going from regular users to Verified Account users. This high value is only due to interactions going from regular users to high profile users. For this reason, in directed graph, generally those nodes have an high number of indegree and a very low number of outdegree. Secondly, when exploring the graph and looking for interesting interactions, like those in which both users contributed to increase the edge value, they should be simply detected by exploring those edges in which the contribute of a node differs for a quantity not going over a certain threshold from the contribute of the other node. This threshold is difficult to define as precise number, but can be found through experiments and a deep insight on all available edges.

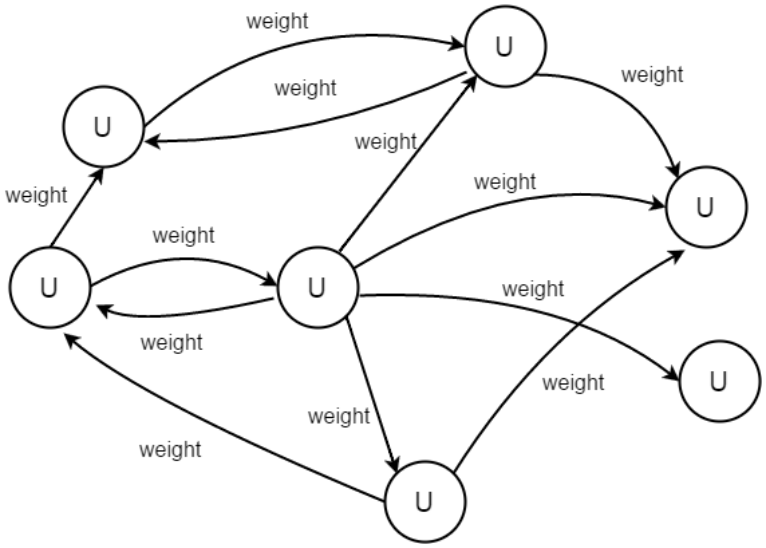

Figure 4.6: *Users Interactions Network*

In Fig. 4.6 it is possible to take a look at an example of what is an interactions graph and how it can be constructed. Clearly the resulting graph may be connected or not depending on how users interacted. Particular cases could be represented by those nodes which have:

- An high outdegree, meaning that the user is particularly active on the platform and looks for interactions with other users, but it may not gain interest in graph analyzers

if other users do not interact with him/her.

- An high indegree, like specified before for Verified Account nodes in Social Media. Also in this case, if this node does not try to interact with other users, this specific characteristics has not a great importance. Or better, it could have it suggesting us to discard this node soon from the analysis.

4.3 Weights Schema Definition

A question is really important now: how to set values regarding the assignment of weights for all interactions related to activities modelled in the *activity network*? This is probably the most significant question in this field and it still represents an open challenge. Till now, there are only few proposal in literature of assigning in an analytical way those weights. Since few papers, book or articles talk about this problem, working on it represents a open research field and solutions are welcome.

The problem of defining weights values comes from the necessity of ranking interactions with the respect to their intrisic importance. If we analyze a tweet as shown in Fig. 4.7 we can see that ways to interact are different and each of it requires a different effort.

Figure 4.7: *Tweet Example*

Looking at the tweet we can see that many people retweeted the content of the author and

it simply means that they agree with him, letting other users be aware of that information. People who comment is much more involved since they would like the user read that comment and preferably answer. If the author had mentioned another user in that tweet, the connection would have been higher and higher since he declares that he is interested in letting that user know about the content. As we can see interactions have different meanings and each of them requires a particular analysis, but it is possible to define a ranking based on importance among them.

After reasoning about many possible solutions to this problem, we defined four possible approaches and we'll explain pros and cons.

- One possible approach could be represented by asking to domain experts to list all interactions in order to define a ranking between them. Domain experts, exploiting their knowledge on the interested social media, should be able to define the ranking explicitly. This approach does not provide a concrete weight schema since as result we only obtain an interaction ranking. As researchers, we should be able to define those weights.

- A solution proposed by [37] is to look for a group of people which we rely on, called "Control Group", that can ensure, an high number of interactions and are familiar with social media. This analysis was done for a social media, Facebook, and it also relies on the fact that all those users know themselves and each of them can give a serious interpretation of those interactions. They are asked to rank interactions from the stronger to the weaker according to the priority they set for each interaction. This phase established a list of values we can use to try some experiments on crawled data.

- Another method can be represented by counting frequencies of interactions between two specific users. The kind of approach proposes to: gathering the data from the social media in a random fashion, counting the number of those activities generating interactions we are interested in (citations, shares, comments), calculating the frequency of each interaction, defining the inverse of frequency as weight for our system. This solution can be refined if the frequency is calculated on more than one data set: this will create many samples thanks to which we able to generate a mean frequency value and consequently a weight schema.

- Last approach provides a solution in which the network analyst decides a ranking between all interactions and decide a schema weight based on particular decisions taken according to the application field in which this study has to be carried out.

The method we want to propose for our work is based on the third approach in the list. It is probably the most analytical one and, even if it cannot reach a precise estimation of those values, it gives a proposal that can be refined. In order to define weights values for direct interactions we gathered three data set in a random way from Twitter. To gather data in a random way, we used the Streaming API to get tweets produced by users without specifying a particular topic. Each data set contains about 50000 tweets. The first operation was to count the number of occurrences of reply, retweets and mentions, discarding, in the last case, all tweets considered for the first two interactions. The reason of this choice is that replies and retweets, when downloaded, already contains mentions, in the text field, that are related to the user who received the retweet or reply action.

After counting the number of occurrences of those activities, we calculated the mean value as it is possible to see in Tab. 4.1. The mean can be used to calculated the frequency of each activity. If we define as M_i the mean value of the occurrences of each interaction i and as M_d the mean values of tweets crawled, the frequency for each interaction is the ratio between this two values.

$$F_i = \frac{M_i}{M_d}$$

The weight w_i related to each interaction can be calculated as the inverse of the frequency.

$$w_i = \frac{1}{F_i}$$

This is the proposed method for defining direct interactions weights. To calculate indirect interactions the computation becomes a little bit more difficult in technical and mathematical fields. Technical difficulties derives from the fact that tweets downloaded for the previous experiment are not good since does not contains values for reply or retweet count.

	Data set	N° Mentions	N° Replies	N° Retweets
	48027	2952	8661	18835
	45620	2698	7323	18286
	52082	3501	8047	21581
Mean	48576	3140	8010	19567
Mean per tweet		0.064	0.164	0.402
Weight		15.47	6.064	2.482

Table 4.1: *Direct Interaction Weight Definition Experiments*

This is due to the fact that the Streaming API retrieves tweet not yet they have been produced and it is impossible to have numbers different from zero in those fields. To perform these calculations we crawled data through the Search API, maintaining the randomness of data sets. We evaluate the sum of these three interactions on all tweets, then we count the number of those tweets and we estimate the mean value. For all previous calculations we considered only tweets having counting fields greater than one since indirect interactions arise only at that moment. To better understand, an indirect interaction can be mapped only if 2 or more users are involved in a specific activity. If two users replied to a tweet, it is possible to map interaction between them, otherwise the only interaction to map is the direct one. All these values are collected in Tab. 4.2 and 4.3. At this point we calculated the mean value for each interaction per tweet. If we call this value M_i where i is related to each interaction, this value will be:

$$M_i = \frac{F_i}{N_i}$$

Data set	$\sum$ Mentions	$\sum$ Replies	$\sum$ Retweets
32208	3337	5033	16758
27215	2172	25	30581
127521	6750	5309	31630

Table 4.2: *Indirect Interactions Occurrences*

Data set	N° Mentions	N° Replies	N° Retweets
32208	1255	850	1573
27215	825	12	3205
127521	2645	1795	6638

Table 4.3: *Indirect Interactions Tweet Count*

where N_i stands for the count number for each interaction i as shown in Tab. 4.4. At this moment we have three sets, related to each dataset, comprehending three mean values, one per interaction. The best solution to aggregate these three values per interaction is to calculate the mean value in order to get a mean value for each interaction per tweet on 3 experiments, called T_i. We may explain these proceedings thanks to the fact that we exploited three data set and we wanted to generalize the way to reach indirect interactions weights. T_i can be defined as

$$T_i = \frac{\sum_y M_y}{D}$$

where M_y is the sum of a specific interaction on the y^{th} data set and D stands for the number of utilized data sets as shown in Tab. 4.5. This value T_i clearly is not an integer and our choice was to round it to the the largest integer that is less than or equal to T_i. The reason for this can be explained since we will never have a floating point number of interaction or a number higher than this. Once rounded, this value has to be used to calculate the actual number of indirect interactions, that is equal to $I_i = T_i * (T_i - 1)$, where I_i stands for the number of interactions for interaction i. To calculate the weight, the formula is:

$$w_i = \frac{1}{I_i}$$

Data set	$F_{Mention}$	F_{Reply}	$F_{Retweet}$
32208	2.659	5.921	10.653
27215	2.633	2.083	9.542
127521	2.552	2.958	4.765

Table 4.4: *Interaction Mean per tweet*

	Interaction Mean on 3 experiments	Number of interactions per tweet	Weight
Mention	2.614	2	0.5
Reply	3.654	6	0.167
Retweet	8.32	56	0.018

Table 4.5: *Indirect Weight Schema*

To sum up, the schema we propose to create the interaction graph is shwon in shown in Tab. 4.6. Adopted values are scaled in a range of values between 0 and 1.

The most valuable interaction in the table is represented by *mention*, followed by *reply* and *retweet*. We were aware of reaching these values and a ranking like this, thanks also to domain experts advices, but all weights shows also that the probability that user A decides to mention user B is more and more lower the other two. This comes from the instrinsic meaning of the mention: on Twitter mentioning user B means that user A really believes that the shared content strongly involves user B and wants to be sure that he/she is aware of

Interaction	Value	Adopted Value
Mention	15.47	1
Reply	6.064	0.392
Retweet	2.482	0.16
Indirect Mention	0.5	0.032
Indirect Reply	0.167	0.011
Indirect Retweet	0.018	0.001

Table 4.6: *Interaction Weight Schema*

it. Instead when user A replies to user B, he/she simply wants to express a thought on what user B is sharing, but user B is not so involved in that comment and he could consider that comment not so important. The power of retweet action is even lower because retweeting means simply sharing a content another user has previuosly produced: this action, in many case, could be tied to the content and to the fact that a specific user shared it. We are also able to say that the retweet activity is approximately 6 times less interactive than the mention one and the reply activity is approximately 3 times less interactive.

In this analysis we decide to give importance also to indirect interactions described in the previous section. Clearly their weight is not so high since the ratio with the direct mention interaction is of an order of magnitude equal to 30 for indirect mention interaction and even higher for the others.

This proposal regarding a weight schema was done thanks to random data set downloaded from Twitter. Even if we rely on Twitter API, we could never be sure of those values and what we want to underline is that a little bias in data sets or in the search we performed could change drastically those values. To be sure of the correctness of ore results we questioned some domain experts which confirmed approximately the veracity of our weights.

5. Analysis

5.1 Architecture Overview

The architecture we wanted to provide in order to perform steps we described in the previous section is based mainly on the chapter reagarding Technologies. One of the fundamental element in this infrastructure is the Twitter Crawler.

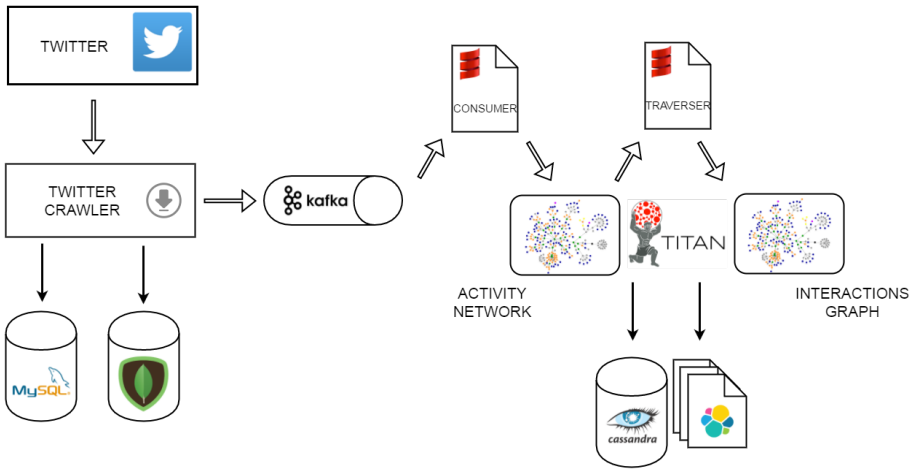

Figure 5.1: *Architecture*

The Twitter Crawler is a software component that is able to download tweets which are related to specific searches a user is interested in. It has the possibility to use the Streaming API, capturing only tweets that are produced from the the starting moment on, or also the Search API, which retrieves tweets starting from the beginning moment back for a week. When downloading tweets from Twitter, it stores data in a MySQL Database and also, if activated, in MongoDb in order to have a copy that can be recovered if problems arises on MySQL. This crawler has been extended in order to prepare messages and to send them to Kafka. We have decided to encode messages in JSON - JavaScript Object Notation - so that they can be parsed in a simple way. A typical message sent to Kafka has a structure

containing 6 main blocks as tweet, user (who produced the tweet), list of entities (contained in the tweet), targetTweet (original tweet that has been retweeted), targetUser (who produced the targetTweet), list of targetEntity (contained in the targetTweet). If the tweet is not a retweet, all fields related to last three component are set to *null*. If the tweet is a reply to another tweet, all available information is stored in three values specified in the tweet block. To send this message to Kafka, Twitter crawler writes this message in a Kafka server through the HTTP protocol. Since Kafka is an asynchronous infrastructure, data remains there till the moment in which a consumer, a program developed with the aim of collecting data, gets messages and starts parsing them. The design of our architecture provides consumers as streaming processes that receive messages not yet they have been downloaded from the Twitter Crawler and sent to Kafka.

Consumers are developed in Scala and, after parsing the message, starts creating nodes and edges in Titan Db following the model described above. Scala Consumers are implemented in a multi-threaded environment in order to execute writing operations quickly. In order to write on Titan, they require a connection to the database and a traversal object able to perform reading operations, useful to avoid adding the same node many times. Titan stores this graph, that represents the *activity network* and allows the traverser program to inspect it in order to look for interactions. Titan relies on a distributed database, that in our case is Apache Cassandra, and on a Index Storage Backend, that is Elasticsearch. Technical features are described in the related chapter.

The traverser program is a single thread program that goes through the *activity network* and writes in Titan the *users interactions graph*, modelling it as described above. It has been developed in a general-purpose fashion in the sense that it can look for different interaction graphs. It is able to generate the interaction graph based only on *mention* activity, *reply* activity, *retweet* activity, pairs of these, etc. Performed queries can look for all tweets containing a particular entity (e.g. we may be interested in all tweets containing the hashtag #flower) or in all tweets containing a specific word (e.g. we may be interested in all tweets containing the word "flower"). The initial step is to set the initial position of the traverser object: in queries interested in an entity we put the traverser in the entity node and then we move it to all tweets nodes containing that specific entity; in queries looking for a specific word, it is placed directly in tweets nodes containing the specified word in their text field. Next steps are shown here and repeated for each of those tweets:

1. We retrieve the user U_i who produced tweet t_i.

2. We retrieve all users mentioned in tweet t_i and we collect them in a list L.

3. We map direct interactions between user U_i and each user contained in L, indirect interactions in all possible pairs of users in L.
4. If t_i has an edge labeled as *reply*, we move the traverser object to the replied tweet r_i.
5. We map a direct interaction between U_i and the user who produced r_i.
6. If r_i has other incoming *reply* edges, we list all those tweets in a list M and we retrieve, for all tweets included in M, the user who produced the tweet. We create a list of those users called R.
7. We map indirect interactions between U_i and all users in R.
8. Above steps 4, 5, 6 & 7 are repeated for *retweet* edges.

After performing all these steps, the traverser program writes the *interaction graph* in Titan and let it be ready for the analysis phase.

5.2 Analysis Overview

Generated Users Interactions Graph shows in a simple way how users interacted in a particular social media after crawling data related to something a researcher is interested in. The analysis of this graph can be performed through a straightforward early practice that consists in looking at it in order to discover interesting relationships among users. The first thing a researcher is able to see is that a few number of nodes are very connected with other nodes: they represent those Verified Accounts, or not, that are often mentioned, retweetted or replied by a huge number of regular users. The expression "very connected" here denotes a huge number of incoming nodes, or indegree.

Definition 6 *The **Indegree** is the number of head edges adjacent to a nodes. It is the number of incoming edges of a node.*

The interesting part, when considering those Verified Account nodes, is to discover if there are edges coming out of those nodes in order to reach other nodes, preferably not other Verified Account nodes. That should be a clear signal that an important profile interacted with a normal person and it would be interesting knowing the reason. The opposite situation can be found when dealing with nodes which interact a lot with other nodes but without having counter interactions. They represents the so-called *trolls*[38], a particular category of users that continuously produces content on a social media, but most of the times without

a so high importance. They become interesting when they receive interactions from other nodes and possibly start interacting with them. They are characterized by a huge number of outgoing edges, or a huge outdegree.

Definition 7 *The **OutDegree** is the number of tail ends adjacent to a node. It is the number of outgoing edges of a node.*

In the case of both high *Indegree* and *Outdegree*, we have users called *influencers*, described in [39]. *Influencers* can be considered as regular users who, thanks to interesting shared content, were noticed by a huge crowd of other users and "gained" reputation on the social media. For this reason, if in the past they were considered highly active users through mentions, retweets and replies, now can be considered passive users since other users are interested in sharing what they produce and in interacting with them. Generally an *influencer* is tied to a specific topic due to the fact that content of his/her post is targeted.

Since we started our analysis counting the number of outgoing and incoming edges, both having a different meaning, it is important to define the degree of a node, a characteristic that after will help us in mapping our graph to a well defined set of networks

Definition 8 *The Degree of a node of a graph is the number of edges incident to the mentioned node.*

Other analysis for each graph aggregate metrics describing the entire networks or node-specific metrics, which identify individuals positions within a network. One of these parameters is the **Assortativity**, which quantifies the tendency of nodes being connected to similar nodes in a complex network. As stated in [40], the *assortativity* can be calculated through the related coefficient. Let's consider p_k as the probability of a randomly chosen node to have degree k. Now, taken a node reached by one of the outgoing edges of the previous node, its degree distribution will be proportional to kp_k, since it is biased by an high degree node, which have an high number of incident edges with respect to others. Let's now define the *remaining degree* q_k as the number of edges leaving the node except for that edge through we arrived. Since the degree is one less than the real degree value, it will be distributed as $(k+1)p_{k+1}$. Hence, q_k will be distributed as:

$$q_k = \frac{(k+1)p_{k+1}}{\sum_j jp_j}$$

Now considering the joint probability distribution of the remaining degrees of the two nodes connected by a randomly chosen edge and call it e_{jk}. Being a probability,

$$e_{jk} = \sum_{jk} e_{jk} = 1,$$

$$e_{jk} = \sum_{j} e_{jk} = q_k$$

and considering $\sigma_q^2 = \sum_k k^2 q_k$ - [$\sum_k kq_k]^2$ as the variance, the assortativity coefficient can be calculated as:

$$r = \frac{1}{\sigma_q^2} \sum_{jk} jk(e_{jk} - q_j q_k)$$

A discussion on the *assortativity* will be presented in the last paragraph of this chapter. Another parameter to take into consideration is **Betweenness centrality**, which is an indicator of a node's centrality in a network. It is equal to the number of shortest paths from all vertices to all others that pass through that node. A node with high betweenness centrality has a large influence when moving information through the network, under the assumption this information follows the shortest paths. Borgatti in[41] states that betweenness centrality focuses on "the share of times that a node i needs a node k (whose centrality is being measured) in order to reach j via the shortest path". In this meaning, k can be considered as "bridge" and betweenness centrality as a measure of how much removing k would break connections between other nodes in the network. Its formula is:

$$C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} is the total number of shortest paths from node s to node t and $\sigma_{\mathrm{st}}(v)$ is the number of those paths that pass through node v.

Closeness centrality too indicates the influence of a node on the entire network, measuring the average distance between a node and every other node in the network. A high closeness centrality means that a node can reach most other nodes in the network in a few number of hops. Conversely, nodes in very borderline spots may have low closeness centrality values; this value indicates the number of hops they need to take to reach distant other nodes in the network. Closeness can be defined as the average length of the shortest path from one node to all other nodes [18]. The mathematical expression for closeness centrality is:

$$C_C(n_i) = \frac{1}{\sum_{j=1}^{g} d(n_i, n_j)}$$

where $d(n_i, n_j)$ is the shortest path between node n_i and n_j.

Last important value we will use is **Density**, defined as the number of edges in the graph with respect to the maximal number of edges. It is a quantitative way to capture important sociological ideas like cohesion, solidarity, and membership.

We will provide an overview of **Cliques**, defined as a subset of nodes of an graph having the induced subgraph complete. The induced subgraph is graph connecting those nodes in a larger graph and it can be considered complete when, taken a random node, it is connected to all others nodes through an edge. In other words each pair of nodes is connected by a unique edge. In the following analysis we won't provide algorithms to look for cliques, but we will discover them only by inspecting the graph. We'll inspect for those cliques avoiding to consider it as a directed graph.

While being interested in the creation of the users interactions graph, it could be important to see if it is possible to compare the graph model we obtained with some of the most important network model in literature. Albert Barabasi in [42] proposes his network model which represents a good approximation of human behaviour in Internet, in the world-wide-wide and in social networks. His model proposes the creation of a random scale-free networks characterized by a preferential attachment mechanism. A network can be defined *scale-free* when the related degree distribution can be compared, at least asymptotically, with a power law function. A *power law* is a functional relationship between two quantities, where a relative change in one quantity results in a proportional relative change in the other quantity, independent of the initial size of those quantities: one quantity varies as a power of the other. Considering the Barabasi model, it means that the set $P(k)$ of nodes having k connections, when k becomes high, goes as

$$P(k) \sim k^{\gamma}$$

with generally $2 < \gamma < 3$. The *preferential attachment* mechanism can be expressed like a behaviour regarding new elements joining a existing set already behaving in a defined way. New elements are more likely to behave as those elements, instead of joining isolated elements or few connected. In Barabasi model *preferential attachment* means that the more connected a node is, the more likely it is to receive new links. Nodes with higher degree have stronger ability to grab links added to the network. When an arriving node has to join the network, the probability to join an *hub* - denoting a very connected node - will be higher than the one to join isolated nodes or *hubs* with few connections. As said above, the characteristic of the Barabasi network model is that the degree distribution follows the power law.

Another network model is the *Watts & Strogatz Model* that produces graphs with small-world properties, including short average path lengths and high clustering as described in [43]. The small-world property of a graph states that even if nodes are not neighbours of one another, it is possible to reach, starting from a node, most of the other nodes by a small number of steps. The most important characteristic of small-world networks is that the distance L between two randomly chosen nodes grows in proportion to the number of nodes N in the network:

$$L \propto log\ N$$

Apart their importance, this model has the limitation of producing an unrealistic degree distribution. It has the limit of not describing well random networks as Barabasi model do.

The *Watts & Strogatz Model* derives from the Erdős-Rényi model, probably the most ancient random network model ever described. In literature this model appeared first in 1959 [44] and then in 1961 [45]. Its description in [46] states that all graphs on a fixed node set with a fixed number of edges are equally likely. When a new node wants to join the graph the probability of connecting to each of the already present nodes is equal. Moreover, each edge has a fixed probability of being present or absent, independently of the other edges.

The characteristic of these two model is that the degree distribution is not represented by the power law, but it clearly follows a random distribution that can often be the poissonian one.

For the analysis and calulcation of all these parameters we use NetworkX[47], a python library providing many methods to calculate all parameters listed above. We develop a python script able to use those methods. Graphs shown in next sections may contain Twitter IDs as user identifiers. This problem is due to the fact that Twitter does not provide the Screen Name of mentioned people. In order to avoid wasting too many API calls to retrieve those Screen Name, we provide IDs. To translate them in Screen Names, this website [1] should be used. Graphs shown below are generated considering the 99^{th} percentile on the edge value. This choice was done in order to avoid presenting very messy graph and to cut those edges that are not representative like those mapping only indirect interactions. In all these three case studies we will present two layouts for the interaction graphs, since both are able to detect important features on users. Something that is not clear in one graph may be clearer in the other.

[1] https://tweeterid.com/

5.3 CasaPound public demonstration Case Study

CasaPound case study refers to a public demonstration organized by the corresponding Italian right-wing party on Saturday 21$^{\text{th}}$, May, 2016 in Rome. The event was very followed in Italy and on Twitter one of the topic trends in that period was signed by the hashtag *#casapountnotwelcome*. We gathered 6271 tweets through our crawler from Thurdsay 19$^{\text{th}}$ to Saturday 21$^{\text{th}}$ May specifying *casapound*, *casapoundnotwelcome* and *forzanuova* (another Italian right-wing party) as words to search. Graphs shown in this paragraph are related to all tweets containing the word *casapound* in their text message and map interactions based on all three activities.

Parameter	Value
Number of nodes	2573
Number of edges	145509
Average in degree	56.5523
Average out degree	56.5523
Density	0.02198
Assortativity	0.26224
Max Edge Value	18.165

Table 5.1: *Casapound Graph Metrics*

Parameter	Value
Number of nodes	411
Number of edges	529
Average in degree	1.2871
Average out degree	1.2871
Density	0.00313
99$^{\text{th}}$ percentile	1

Table 5.2: *99th percentile Graph Metrics*

The graph in Fig. 5.6 is preferable when we are looking for those users that are central in the graph and have high indegree or outdegree. One of this users is the one with Twitter ID: 990052896 related to the account of *Simone Di Stefano*, a member of CasaPound party. Thanks to this graph and analyzing his profile we were able to know that he was candidate for becoming mayor of Rome in June 2016 election. Other important users in this sense is the Official page of *@CasaPound_Italia*, *@difendereroma* (page of another member of this party), *@CircoloFuturista* (page related to something commemorating Fascists ideas) and so on. Clearly choosing a political topic gives the way to opponents for sharing content against this party and this demonstration. In fact a very particular profile is the one of *@valibona44* clearly visible in the right side on Fig. 5.7. Exploring edges coming out from this node we were able to find the Twitter Account of *@vinzveg*, a user that, in the same day in which CasaPound party was manifesting in Rome, shared a video in which it is possible to see a manifestation against CasaPound. The user *@misscheeky666* did the same and, in the following days, shared other photos and videos of antifascist demonstrations. Clearly our intent is not to report

these profiles to competent authorities, but it is remarkable that the usage of an theoretical instrument like this could avoid the birth of unpleasant situations, given the hostility all over the world between right and left side political members. Consider the case in which those two demonstrations reached the same place and started communicating, surely in a wrong way. Police should have been there to monitor the situation and avoid problems to both groups.

After this brief visual analysis of the graph, we can now analyze Betweenness values. Among the first twenty users having an high Betweenness value, apart those users related to members of this Italian party and some journalists, there are some very particular profiles. One of these is *@ervenrossetti* (Betweenness value: 0.00076), a user that clearly declares himself fascist, *@bloodaxes1* (Betweenness value: 0.0002), someone evoking the nazist period and the purity of the Aryan race, and not surprisingly *@valibona44* (Betweennes value: 0.00005), which we were able to inspect thanks to a visual search. As a counterpart we can find also *@tulliocampana* (Betweenness value: 0.00001), representative of an opposed Italian Party. Since those users have the highest values for Betweeness Centrality, they can be considered key-nodes for the information flow. The Closeness Centrality, in the first twenty users, apart presenting many users already present in the Betweenness ranking, underlines the presence of *@pissi_marini* (CC value: 0.03749) and @MarcoCapoccetti. After looking at their profile, these users can be considered of completely opposed political party. However, having so high values for Closeness Centrality, they can be considered in *influencers* since in this interactions graph many users are connected to them.

Once the analysis of users is completed, the important issue to remark is the discussion on the network model present in this graph. In Fig. 5.2 we are able to see the degree distribution of the original network. In the following images we have the degree distribution of networks generated through Barabasi, Erdős-Rényi and Watts-Strogatz model. It is clear that the the degree distribution of our network is similar to the one of the Barabasi Network. This characteristic can be explained this way: let's consider a node joining the network that, in social media environment, can be compared to a user willing to interact with other users. The probability that this user will retweet or reply a tweet of a Verified Account profile or the probability of mentioning a political candidate to express his opinion should be higher than the one of mentioning a common user to speak about political issues. In a more general sense it is more likely to interact with a Verified account than with a regular user regardless of the topic. This explains the typical phenomenon of preferential attachment of the Barabasi model. The degree distribution of our graph presents some cases in which the trend in the plot is not really equal and we have the presence of outliers. This is due to the fact that we are operating on real data, not on ideal data.

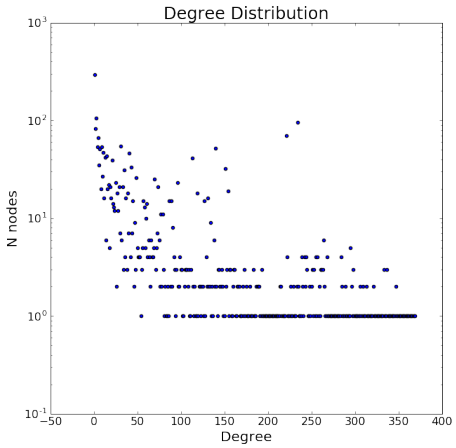

Figure 5.2: *Casapound Degree Distribution*

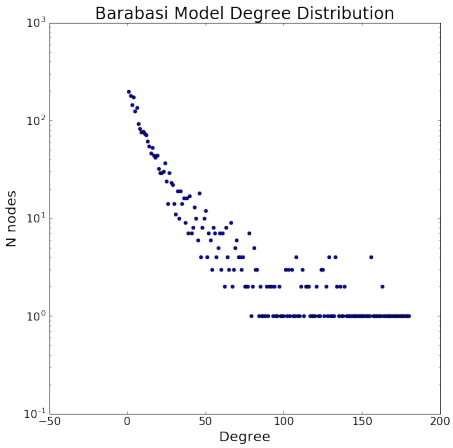

Figure 5.3: *Barabasi Degree Distribution*

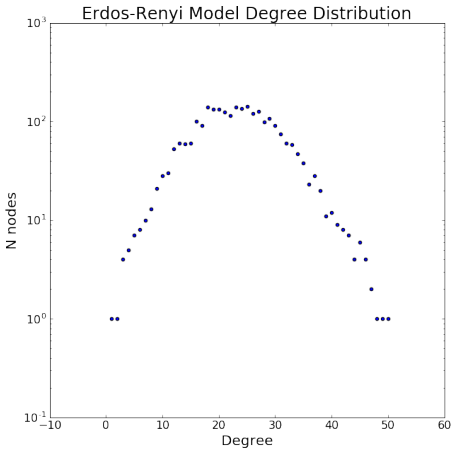

Figure 5.4: *Erdős-Rényi Degree Distribution*

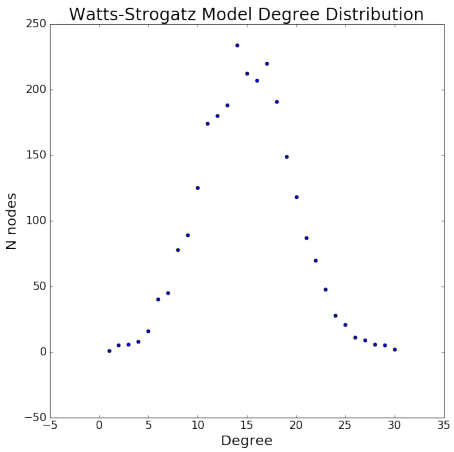

Figure 5.5: *Watts-St. Degree Distribution*

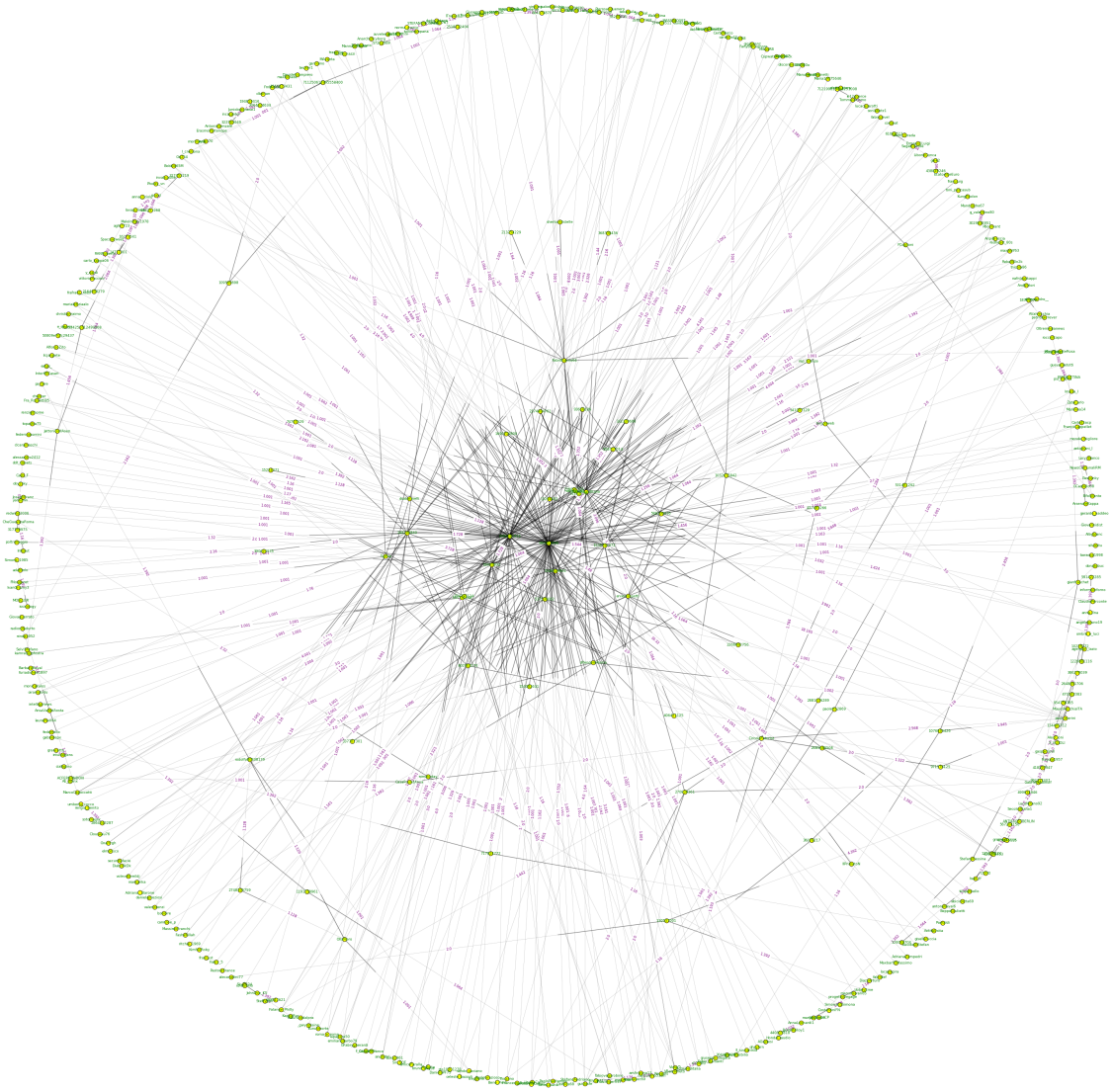

Figure 5.6: *Casapound Fruchterman Reingold Layout Graph*

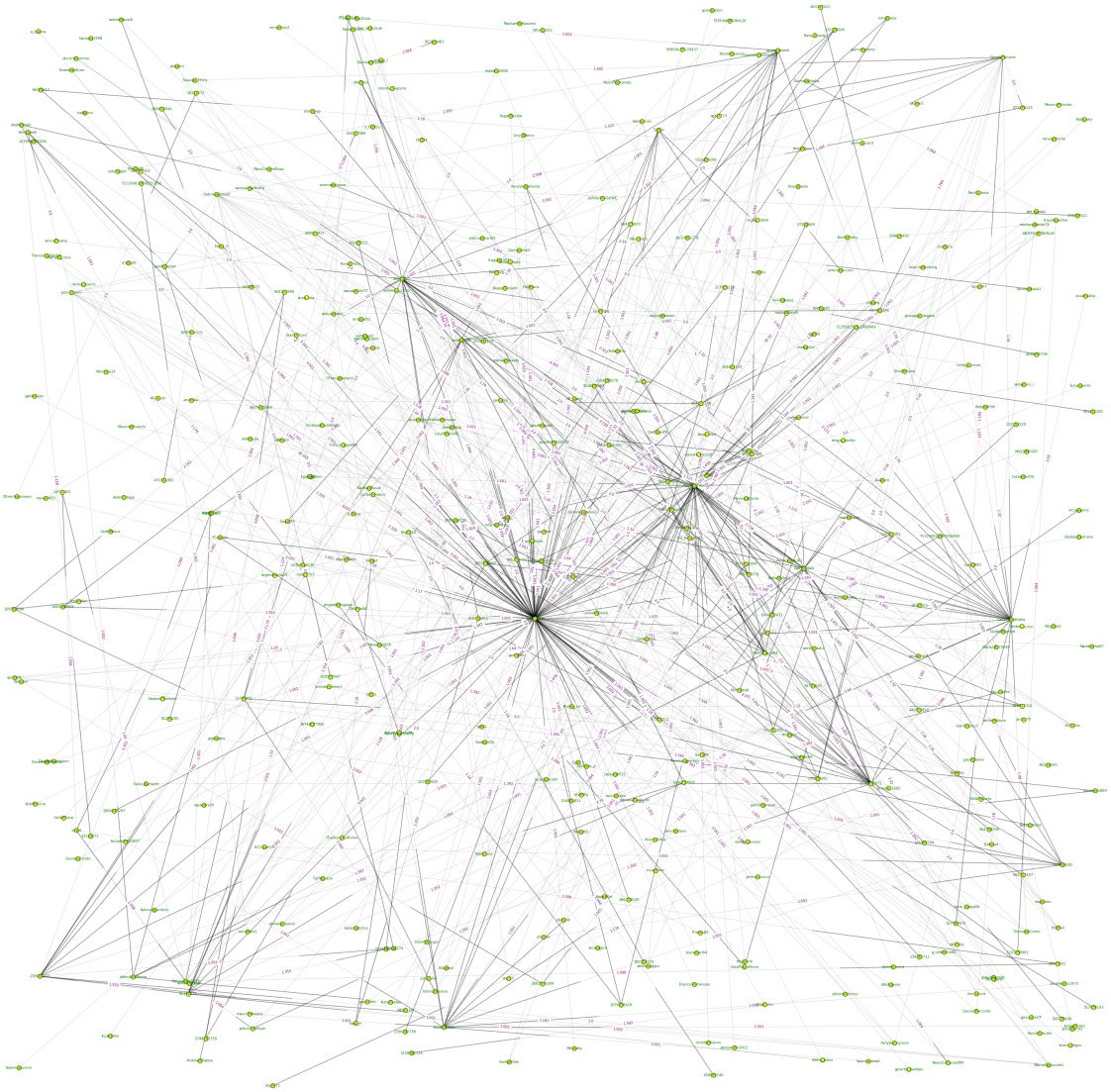

Figure 5.7: *Casapound Random Layout Graph*

5.4 Marco Pannella death Case Study

This case study is based on the event related to the death of Marco Panella, an Italian politician of radical party. He died on Thursday 19$^{\text{th}}$, May, 2016 and we crawled 24072 tweets in only 4 hours in the same day. Words specified in the search were *ciaomarco*, *marcopannella* and *pannella*, trending topics on Twitter in that day. Graphs shown below are related to all tweets containing the word *marcopannella* in their text message and map interactions based on all three activities.

Parameter	Value
Number of nodes	6094
Number of edges	244829
Average in degree	40.1754
Average out degree	40.1754
Density	0.00659
Assortativity	0.06968
Max Edge Value	16

Table 5.3: *Pannella Graph Metrics*

Parameter	Value
Number of nodes	506
Number of edges	596
Average in degree	1.1779
Average out degree	1.1779
Density	0.00233
99$^{\text{th}}$ percentile	1

Table 5.4: *99th percentile Graph metrics*

The graph shown in Fig. 5.9 presents some entities that are not visible in other case studies. Taking a look at the center of the graph we can discover some cliques. These cliques mainly contain three nodes users, *@Paolomasi72Li* connected to *@SaveTibetOrg* and *@MatteoMecacci*, which are connected themseves too. Another three cliques can be found in the top left corner where *@SenatoriPD* and *@mbolognetti* are connected themselves and with respectively *@Beatrix_aka_BM*, *@elio_sileno* and *@fedefresa*. In this case users are not so important if we consider their social aspect, they are all members of an Italian political party interacting with the account of senators of that party. Normally cliques are important since are able to group people. Cliques form themselves when users are interacting on a particular that can be considered interesting only by them. For this reason cliques, if mined, are in charge of showing the specific topic and, maybe, the reason of that discussion. Generally such users were used to continuously mention and reply augmenting the interaction value between them.

The analysys of Betweenness values does not lead to discovery of so many particular cases as for the previous case study. We found *@RadioRadicale* (Betweenness value: 0.00155), the account of the official Radio of the Italian party headed by Marco Pannella, *@MarcoPannella*

(Betweenness value: 0.00138), that has an indegree due to all tweets in which he was mentioned. Interesting users revealed by the Betweenness value are *@ParzialePaolo* (Betweenness value: 0.00008) and *@ChristianDuDu1* (Betweennes value: 0.00006), which can be considered regular *influencers* users. Top three *influencers* of this network, detected through closeness centrality, are *@Paolomasi72Li* (CC value: 0.01848), *@RobRe62* (CC value: 0.01677) and *@flamanc24* (CC value: 0.01296). These three users, even if they have an enormous number of produced tweets, should be common people who acquired a lot of reputation in this Twitter. The profile of *@Paolomasi72Li* may be monitored since it may be compared to a troll. This user has many outgoing edges, but no incoming edges.

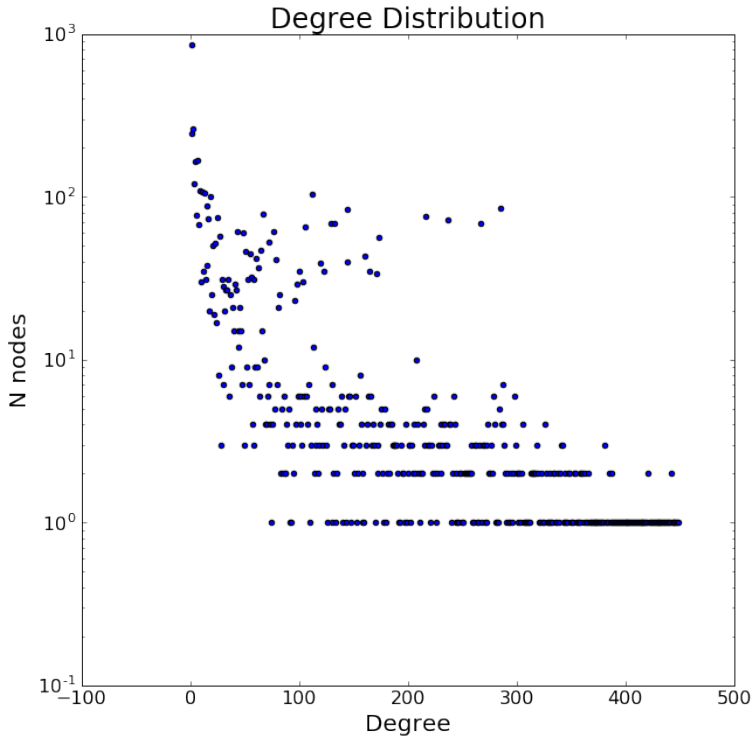

Figure 5.8: *Pannella Degree Distribution*

As we can see from the Fig. 5.8, the degree distribution of the graph follows the power law, as stated for Barabasi model networks.

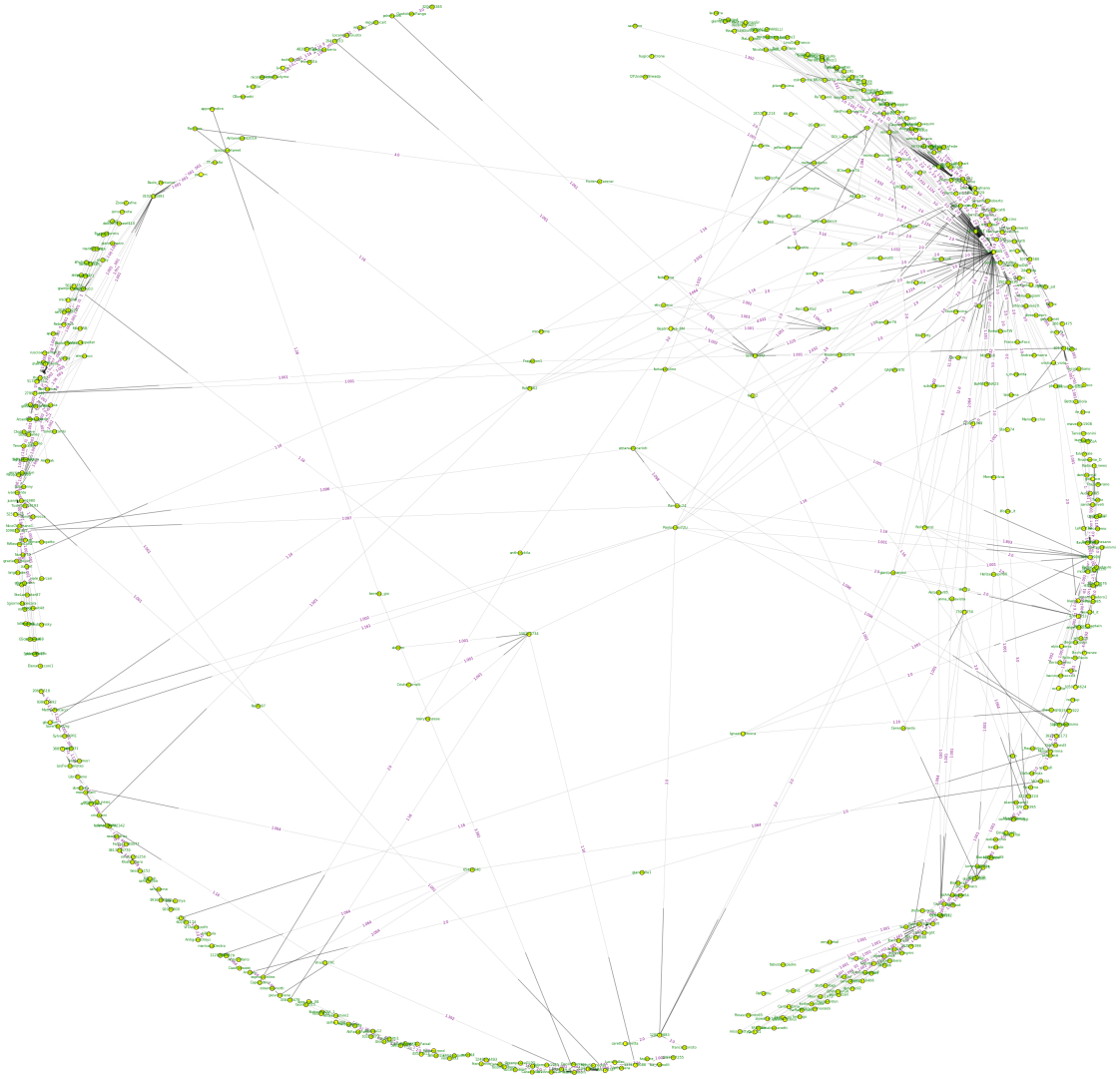

Figure 5.9: *Pannella Fruchterman Reingold Layout Graph*

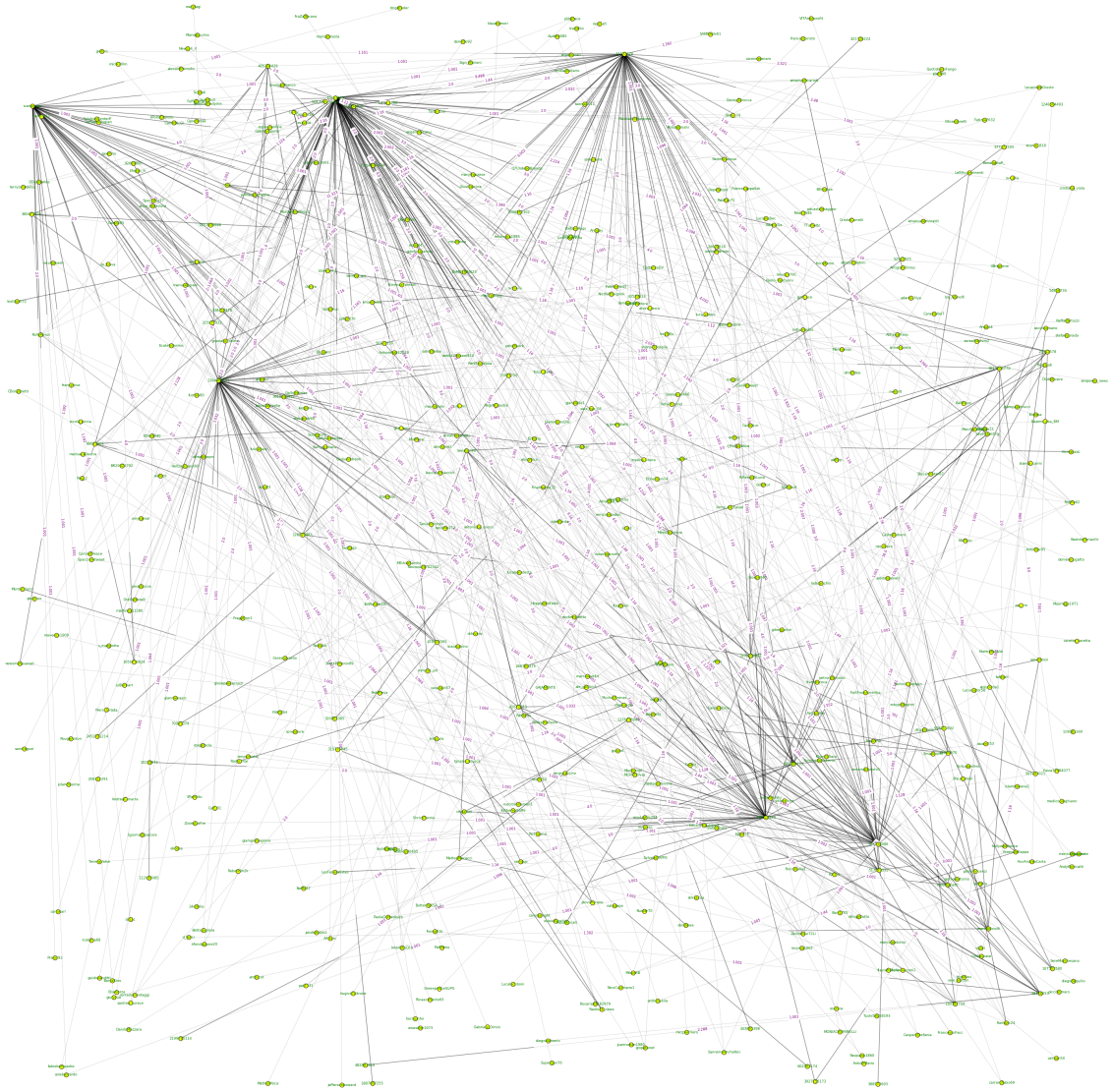

Figure 5.10: *Pannella Random Layout Graph*

5.5 Giro d'Italia 9^{th} Stage Case Study

The 9^{th} Stage of Giro d'Italia gave us the opportunity to study a sport event. In that stage the Italian favored to win cyclist Vincenzo Nibali did not have a great performance and lost positions in the general ranking. Conversely, Tom Dumoulin, another cyclist candidated himself to final victory. We crawled data through the Search API on Wedneday 18^{th}, May, 2016 and we got 21478 tweets. *Dumoulin*, *nibali*, *giro* were words specified in that request. Graphs shown below are related to all tweets containing the word *nibali* in their text message and map interactions based on all three activities. They are not generated considering the 99^{th} percentile as for previous graph, but the 85^{th} percentile.

Parameter	Value
Number of nodes	572
Number of edges	6457
Average in degree	11.2885
Average out degree	11.2885
Density	0.01976
Assortativity	0.5355
Max Edge Value	4

Table 5.5: *Nibali Graph Metrics*

Parameter	Value
Number of nodes	572
Number of edges	815
Average in degree	1.4248
Average out degree	1.4248
Density	0.0025
85^{th} percentile	0.001

Table 5.6: *85^{th} percentile Graph Metrics*

From those two graph in Fig. 5.12 and 5.13 we can see the presence of a lot of central nodes like cyclists *@DiegoUlissi* (Twitter ID: 270928643), *@vincenzonibali* (Twitter ID: 114422199) and *@Andrey_Amador* (Twitter ID: 327585857), cycling teams like *@Movistar_Team* and the official profile page of Giro d'Italia, *@giroditalia*. What is really clear is that a lot of users interacted both with *@DiegoUlissi* and *@giroditalia* since this particular stage was won by this cyclist. Also in sports topic are present antagonist users who interact a lot sustaining one athlete instead of another. Probably this was not the case since hardly all of them supported Nibali.

The top left corner of the graph in Fig. 5.12 point out the presence of a clique between *@jonasvila*, an old Dominican cyclism champion, *@anaaraya74* and *@ivanypaula1512*. Betweennes values for this graph are not able to provide any interesting profile. Besides a long list of cyclists, in the top twenty there are only Italian and foreign journalists or newspapers profiles. The Closeness Centrality shows the presence of many normal people who are *influencers* in this conversations like *@anaaraya74* (CC value: 0.02104) and *@Sheeylaa* (CC

value: 0.02189).

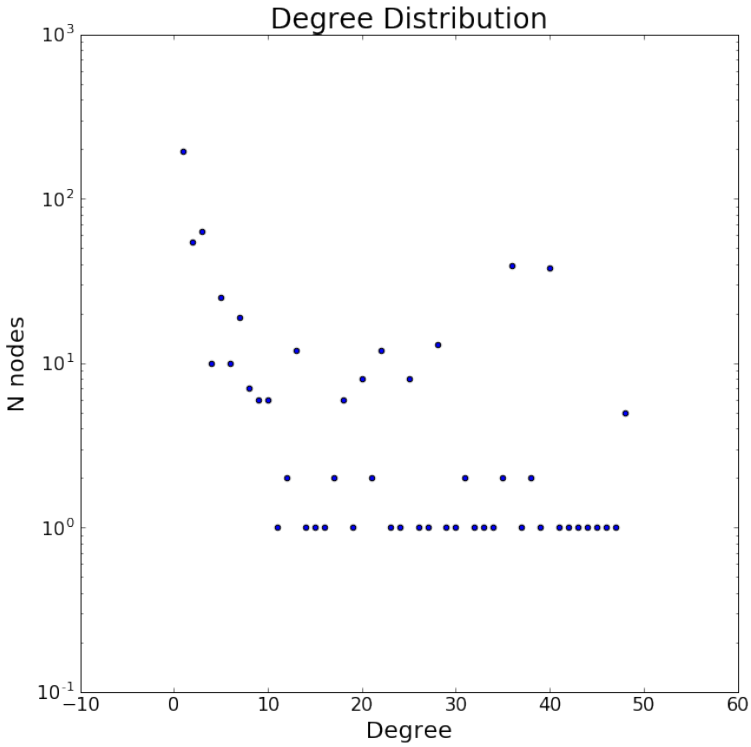

Figure 5.11: *Nibali Degree Distribution*

The Barabasi model is clearly underlined also in this graph, confirming our theory, as shown in Fig. 5.11. The evident aspect that characterizes the concept of preferential attachment mechanism in Barabasi Network, is shown when looking at those nodes who interacted with *@DiegoUlissi* and *@giroditalia*. Supporters, after the end of this stage, wanted to congratulate themselves with the winner and they produced many tweets mentioning both this cyclist and the official profile of Giro D'Italia. They were not really interested in interacting with other users or profiles.

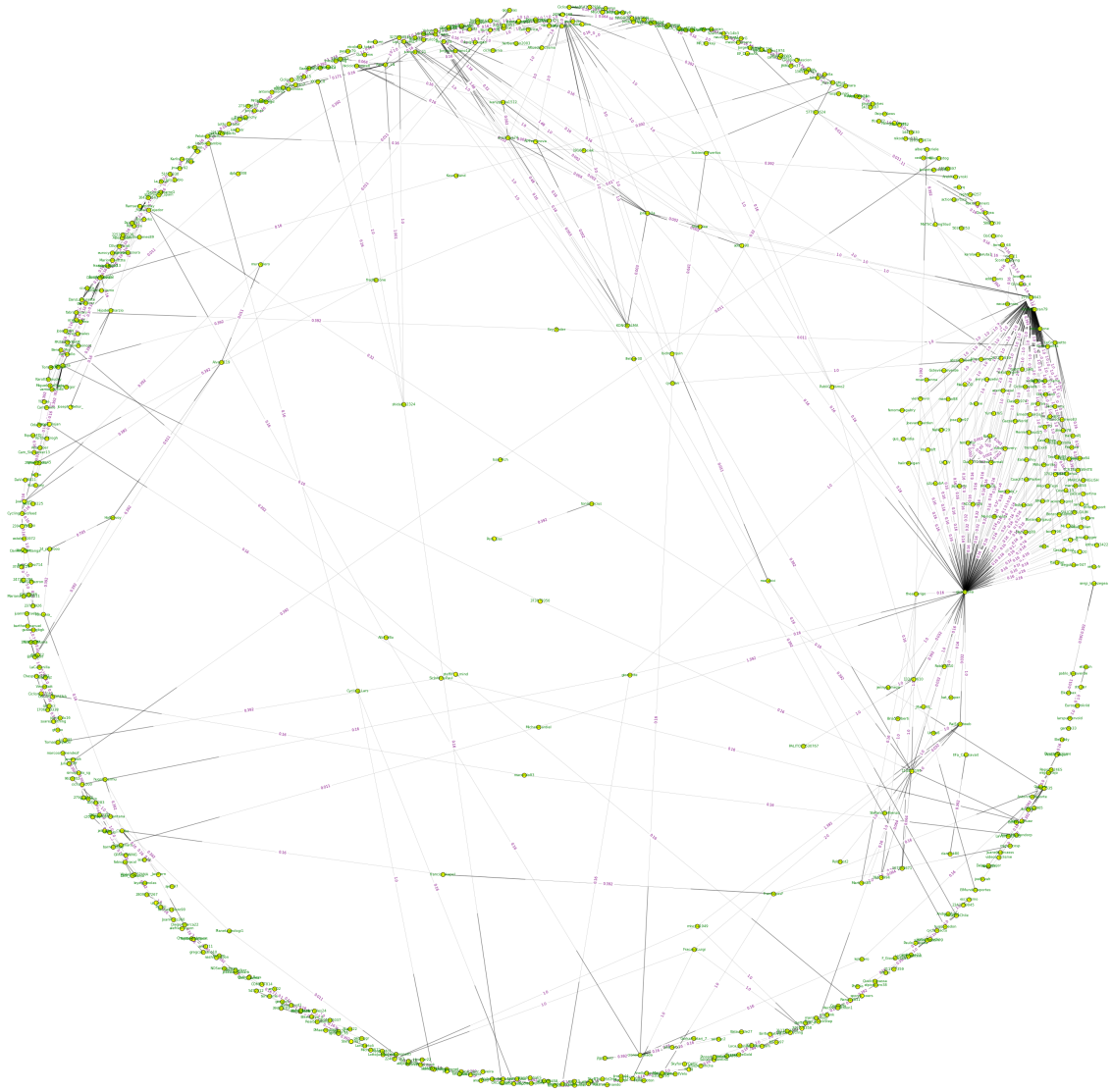

Figure 5.12: *Nibali Fruchterman Reingold Layout Graph*

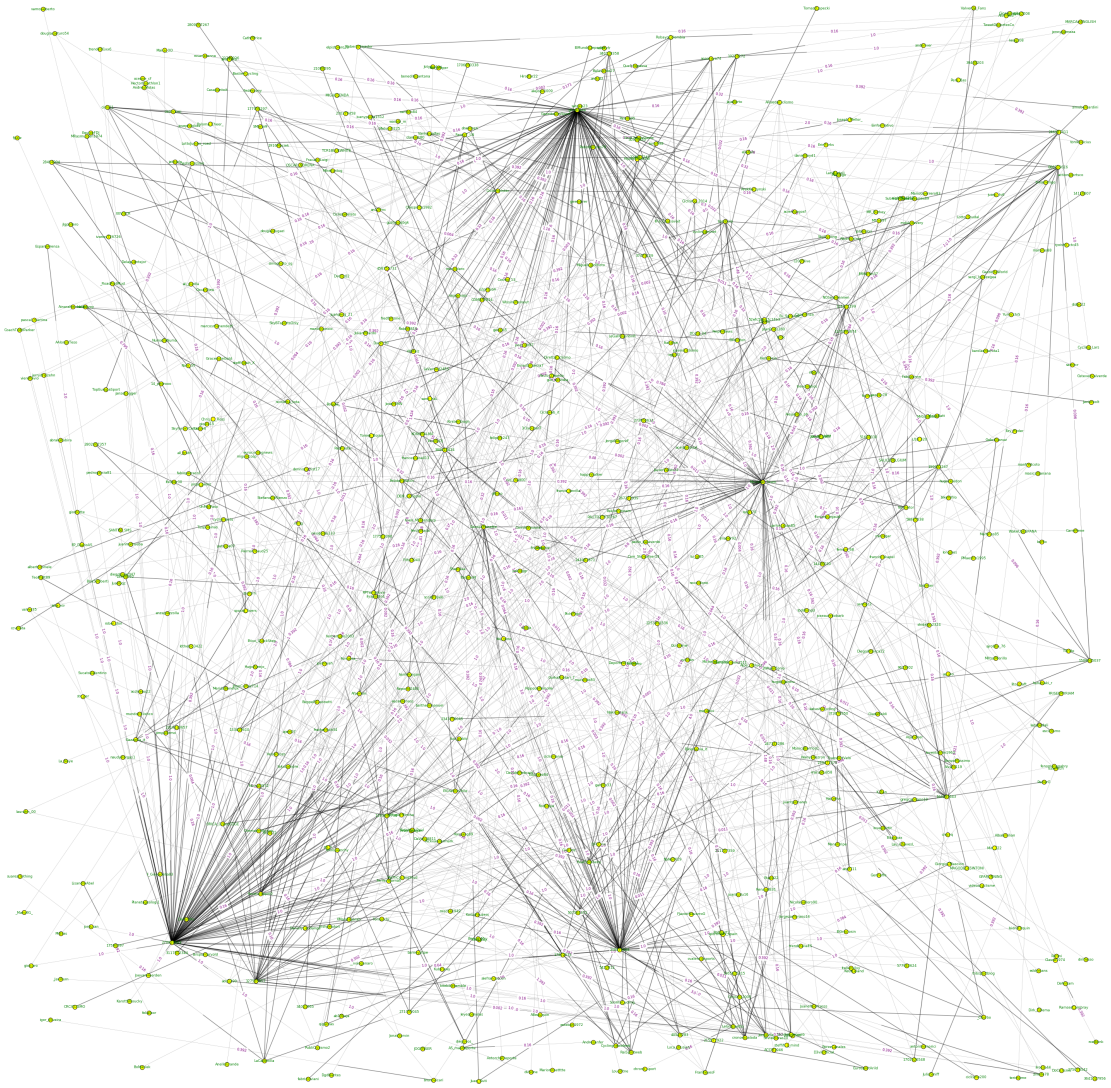

Figure 5.13: *Nibali Random Layout Graph*

5.6 Further Analysis Considerations

One of the metrics defined in the second paragraph of this chapter is *Assortativity*, a metrics revealing that nodes have the tendency to connect themselves with other nodes having similar degree values[40]. For the graph related to Nibali case study this value is equal to 0.5355, that represents a high value for *assortativity*, considering that a subset of different edges values is ever present. For other graphs instead, this value decreases to 0.26224 and 0.06968. Theory on *assortativity* says that, when the number of nodes increases, this value tends to zero in graph created through the Barabasi model. Looking at values in tables related to each case study, we can clearly see that this hypothesis is confirmed in our experiments. Unfortunately, Erdős-Rényi graphs too have an *assortativity* tending to zero when the number of nodes increases and, for this reason, previously we did not mention this property to highlight the similarity with Barabasi model graph.

Another important consideration regards Betweenness Centrality. This metrics, depending on the topology of the *interactions graph*, may reveal if some nodes are *influencers* or not. Consider the case in which a node N_i has an high *indegree* due to the fact that many users retweeted, through a tweet t_j, one of the tweets N_i produced, called t_i. Now, consider also the situation in which other users, interested in t_i, desire participating in this conversation. Those users decide to retweet t_j instead of retweeting t_i. The case is shown in Fig. 5.14 These activities would create a sort of second-level circle of interactions around N_i. This would mean that if we remove N_i, we break all connections among all nodes connected to N_i. This situation establishes that N_i is an *influencer*.

Figure 5.14: *Betweenness - Central node case*

Probably the most important case in which betweenness centrality may reveal *influencers* is the case in which a node connects two very connected users, situation shown in Fig. 5.15. In that case, those users may be considered *influencers* since they are connected to two users who, thanks their high closeness centrality, are considered *influencers* themselves. In a sense, removing that node (e.g. the yellow one in the figure) would break the flow of information between two *influencers*. Situations like this may underline the presence of regular users as *influencers*, even if it may represent a difficult task. The best approach to discover important information from this metrics would be to calculate it for all nodes and then inspect those nodes, having higher values, directly through a visual analysis of the graph.

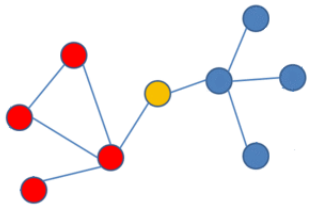

Figure 5.15: *Betweenness - Bridge node case*

6. Future Work

The work presented here surely represents a starting point for researchers who would like to take part in this research field. The work could be improved, a new infrastructure could suitably be designed for experiments like these, a new weight schema could be preferably proposed on more scientific foundations even if this example shows that is hardly impossible to define it this way. In this section we are focused on giving some possible hints to improve this work either in a theoretical direction or in a technical one.

6.1 Social Media View Enlargement

The infrastructure proposed for this work shows a way of creating *users interactions graph* from Twitter data. A program, representing the Twitter Crawler, downloads a bulk of data that is analyzed real-time by a Scala Object. Before this phase, data is sent to a Kafka publish/subscribe architecture that let data be available for more than one consumer interested in processing data. The consumer parses received data and adds some nodes and edges to the *activity network* writing the graph in Titan Db. A traverser goes through this graph and generates a *users interactions graph* expressing the values of how much users are joint in a specific discussion.

The short description above underlines some main points where a researcher could intervene in order to enhance this work. One of these is represented by the possibility to crawl not only a social media but more than one, being able to define more than an *interactions graph* in Titan. The enrichment of social media view requires the development of a crawler per each different social data provider and preferably a social media unified data model suitable for all sources. This states that, even if it's true that generally interactions are different on several social media and their semantic meaning could slightly differ from other interactions, a unified data model able to comprehend all interactions on different platforms would create a milestone standing in front of all works that can be realized through Big Social Data. A researcher, knowing that all interactions provided by a social media can be represented by a

specific unified data model, will be able to map this model on top of data arriving and will generate the interaction graph in a short way. This is perfect when a new social media starts being on the cutting edge, involving researchers to work on it.

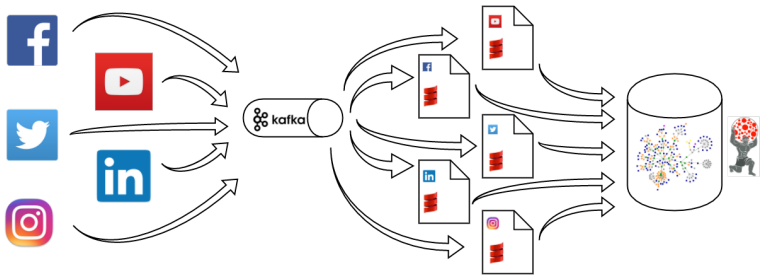

Figure 6.1: *General-purpose Architecture*

Once crawlers are ready to download data and send it Kafka, the approach is to provide a set of consumer parsing those data. Kafka enables producers to store data in different topics and enables consumers to retrieve data from one or more topic. Therefore augmenting the number of social media sources should represent one improvement to this system.

6.2 Cross-Social Users Interactions Mapping

The creation of more than one interaction graph let us believe that the single process of discovering particular information would lead to list pairs of users being very connected on each different social media. An interesting approach could be the one in which we are focusing our attention on all these pairs of users and we would like to discover if two users interacted either in a social media or in another one. This phase requires a very difficult preliminary step regarding the conjunction of profiles coming from a social media with those coming from another one. This step can be simply accomplished when dealing with famous people since they provide Verified Accounts and generally, just by looking at names, the similarity stands out. When facing with regular people, this task requires a careful analysis of profiles since frequently users have different names on various social media and detecting correspondences would require the analysis of all information profiles provide. If this profiles reviewing process reveals correspondences, we will have to inspect all interactions looking for pairs of users who interacted in two or more social media. The strength of this approach would provide a powerful instrument able to state that, on a specific topic, crawled messages reveal that a connection between two specific users is present in more than one social media.

This information clearly defines that the specific topic, regardless of the social media, spurs a user to interact with another specific user.

6.3 Custom Interaction Definition

The functionality of Kafka reveals that data are stored in it and more than one consumer can retrieve data to parse them according to the work that has to be carried out. Kafka allows consumers to be grouped in bunches and each consumer can subscribe a particular topic to retrieve data stored in it. That said, we could think about defining an interaction in many different ways. Till now we defined as interaction an activity on a social media performed by a user towards another, like *tweets, retweets, mentions, etc.*. Since each node contains a lot of properties providing information about the specific represented entity, we should exploit this information to create a particular interaction. Consider the case in which we are involved in a project interested in dealing with text and we are focusing the attention on the detection of hate speech in social media: a deep analysis of tweets text using text mining techniques could be a starting point to define the interaction as a link between two users who were used to produce *hate speech* on a social media. This interaction definition would lead to discover communities of users using an impolite pitch and allowing social media technicians to obscure to other users those words and to help them in apply ethical actions towards those users.

Text surely represents a feature full of interest, but nodes also contains a property that regards a geographical position. Tweets may contain the position in which have been posted, users may specify the city in which live or work. Modelling an interaction through the geographical position would lead the analyst to consider, for example, all tweets about a specific topic produced in a certain area. This information could be used this way: journalists are talking about a political manifestation that is taking place in a certain square and many people are posting about this event; suddenly an accident occurs and police, analyzing the interactions network, could be able to say who are those people posting about this specific event in that specific place and could question them to have better information about that accident.

In the example shown above, we highlight the importance of position in node information. The consequent feature to consider is time, generally always available. Interactions modelled on top of time could be used to understand why a community of users is posting contents in that moment and why they are interacting. A specific event in fact could generate a so high number of interactions able to discover pairs of users talking about something. The election of a new president, a great sports event or breaking news are only some of the examples which

could generate those interactions. Time has another strong property: given a specific topic, we could be interested in crawling it at different moments in a specific time interval. Time in this case could reveal that the interaction between two users could increase, shrink or stay the same over time. We may plot the interaction value over time in order to understand how and why it varies.

7. Conclusion

The past few years have seen a dramatic increase in social media usage with most internet users having at least one account on each platform. Data stored in social media represents an available information that can be used according to our own work. Social Media Analytics is the branch that performs tasks like gathering, storing, analyzing and visualizing those data. New tools and frameworks have been developed to achieve this goal, even if many of them do not rely on efficient and automatic techniques. In order to understand the importance of the information related to users, we have to notice that it is used by many organizations to improve their capabilities. Enterprises and Institutions are thrilled about users information since it can modify their way of acting in order to enhance performances in their work field.

The important aspect of discovering relationships among users on social media can be considered one of the most important research fields. Even if the most common approach, in detecting these relationships, may be seen in inspecting mere friendship connections, an intriguing practice is represented by the study of activities performed by users. In this thesis we wanted to give an answer to these questions: 1) How to model Social Media data in order to generate users information? 2) Which are the steps to perform in order to reach this goal? 3) How to deal with Big Social Data?

These problems have been solved by proposing a model that is able to transform data in a *users interactions graph*: firstly we studied a solution to cope with Twitter Data and to extract information from them; secondly, we defined a model that clearly maps all activities performed by users on Twitter in a graph, called *activity network*; thirdly we exploited this graph to retrieve interactions among users and assigned a weight to each interaction. The *activity network* model contains a complete mapping of Twitter subjects including users, tweets and entities (e.g. photo, video, url, hashtags), represented by "nodes", and a definition of activites such as *tweets*, *retweets*, *reply*, *mention* and *contains*, represented by "edges" connecting said nodes. Starting with the tweets or entities, we are able to list all interactions traversing the graph.

The definition of a weight schema is one of the most difficult steps since there still isn't a

validated approach to define it. Our solution is based on the study of interaction frequency in randomly gathered data sets. Defining a weight schema is crucial for this research field but, since literature up until now has been unable to provide valid solutions to this issue, this is still something we are working on.

The characteristic of our data, that can clearly be considered Big Data, encouraged us to use technologies built in a distributed fashion, like Apache Cassandra or Elasticsearch - two tools that Titan relies on. It is important to note that most of the tools used in this work were implemented by social media themselves: Apache Cassandra was developed at Facebook Headquarters (and used by Twitter too) to substitute the previous database infrastructure, later replaced by Apache HBase; Apache Kafka was developed by Linkedin and replaced their previous message queues. The choice of Titan as Graph database could probably inspire debate on better solutions. Neo4J, OrientDB or DEX are some possible solutions that can replace Titan, even though the latter has the advantage of being open source and recent. Our system can be optimized using Spark Streaming, which is able to distribute work in several nodes, when the *activity network* is created on Titan. As stated in the Technology chapter, the interfacing between Titan and Spark is still under deployment. For this reason, in order to let operations be more scalable, we decided to just use Scala.

Regarding the analysis process performed on *user interaction graphs*, we provided a study of some metrics able to characterize the graph like *average indegree* and *outdegree*, *assortativity*, etc. Other metrics have been exploited as a support to the visual analysis, a step in which we were able to identify particular connections among users, "trolls", most connected users as well as cliques and "influencers". As a result, we observed that *influencers* in these graphs are users with a high closeness centrality value; in some cases they have also high values for betweenness centrality. The direct inspection of some profiles on social media gives an idea as to why/how those specific users play a significant role in the graph. Moreover, even if accounts belonging to newspapers, journalists, celebrities or high profile people are the most relevant, we must also focus on the interactions among regular users. Finally, we proved that the generation of this graph follows the *power law* as indicated in the Barabasi model. We demonstrated this by comparing the degree distributions - this allowed us to confirm that there is a similarity between the Barabasi and our degree distribution. We also confirmed that there are differences between our distribution and the distributions by Watts-Strogatz and Erdős-Rényi. In fact these models have a poissonian degree distribution.

Results show that analyzing activities on social media and generating interactions, instead of considering mere friendship connections, provides us with a more precise way of understanding user behaviour on these platforms. The *activity network* gives us a complete map

of how users behaved and their intentions when dealing with Social Media. The *interactions graph* translates information embedded in the previous network and provides an overview of how users interacted. Moreover, this work confirms two findings: 1) the more users are active and the more they participate on Social Media, the more others are interested in their profiles; 2) even if the model presented in this work relies on Twitter, the redefinition of the *interaction* term and the choice of a different platform may allow the adaptation of this system to other case studies and environments.

We are sure that in future better improvements either in the architecture or in the theory of our work may lead to the discovery of better solutions able to improve its performances.

Bibliography

[1] A. M. Kaplan and M. Haenlein, "Users of the world, unite! the challenges and opportunities of social media," *Business horizons*, vol. 53, no. 1, pp. 59–68, 2010.

[2] K. SMith, "Marketing: 96 amazing social media statistics and facts for 2016," 2016. [Online]. Available: https://www.brandwatch.com/2016/03/96-amazing-social-media-statistics-and-facts-for-2016

[3] J. H. Kietzmann, K. Hermkens, I. P. McCarthy, and B. S. Silvestre, "Social media? get serious! understanding the functional building blocks of social media," *Business horizons*, vol. 54, no. 3, pp. 241–251, 2011.

[4] R. L. Sie, T. D. Ullmann, K. Rajagopal, K. Cela, M. Bitter-Rijpkema, and P. B. Sloep, "Social network analysis for technology-enhanced learning: review and future directions," *International Journal of Technology Enhanced Learning*, vol. 4, no. 3-4, pp. 172–190, 2012.

[5] S. Kumar, F. Morstatter, and H. Liu, *Twitter data analytics*. Springer, 2014.

[6] S. Kaisler, F. Armour, J. A. Espinosa, and W. Money, "Big data: Issues and challenges moving forward," in *System Sciences (HICSS), 2013 46th Hawaii International Conference on*. IEEE, 2013, pp. 995–1004.

[7] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the evolution of user interaction in facebook," in *Proceedings of the 2nd ACM workshop on Online social networks*. ACM, 2009, pp. 37–42.

[8] E. Gilbert and K. Karahalios, "Predicting tie strength with social media," in *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM, 2009, pp. 211–220.

[9] C. Wilson, B. Boe, A. Sala, K. P. Puttaswamy, and B. Y. Zhao, "User interactions in social networks and their implications," in *Proceedings of the 4th ACM European conference on Computer systems*. Acm, 2009, pp. 205–218.

[10] M. Conover, J. Ratkiewicz, M. R. Francisco, B. Gonçalves, F. Menczer, and A. Flammini, "Political polarization on twitter." *ICWSM*, vol. 133, pp. 89–96, 2011.

[11] A. Mackinnon, "A spreadsheet for the calculation of comprehensive statistics for the assessment of diagnostic tests and inter-rater agreement," *Computers in biology and medicine*, vol. 30, no. 3, pp. 127–134, 2000.

[12] K. J. Ottenbacher, W. C. Mann, C. V. Granger, M. Tomita, D. Hurren, and B. Charvat, "Inter-rater agreement and stability of functional assessment in the community-based elderly." *Archives of physical medicine and rehabilitation*, vol. 75, no. 12, pp. 1297–1301, 1994.

[13] M. Cha, H. Haddadi, F. Benevenuto, and P. K. Gummadi, "Measuring user influence in twitter: The million follower fallacy." *ICWSM*, vol. 10, no. 10-17, p. 30, 2010.

[14] R. Xiang, J. Neville, and M. Rogati, "Modeling relationship strength in online social networks," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 981–990.

[15] H. Chun, H. Kwak, Y.-H. Eom, Y.-Y. Ahn, S. Moon, and H. Jeong, "Comparison of online social relations in volume vs interaction: a case study of cyworld," in *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*. ACM, 2008, pp. 57–70.

[16] N. B. Ellison, C. Steinfield, and C. Lampe, "The benefits of facebook friends: social capital and college students use of online social network sites," *Journal of Computer-Mediated Communication*, vol. 12, no. 4, pp. 1143–1168, 2007.

[17] J.-P. Onnela, J. Saramäki, J. Hyvönen, G. Szabó, M. A. De Menezes, K. Kaski, A.-L. Barabási, and J. Kertész, "Analysis of a large-scale weighted network of one-to-one human communication," *New Journal of Physics*, vol. 9, no. 6, p. 179, 2007.

[18] S. Wasserman and K. Faust, *Social network analysis: Methods and applications*. Cambridge university press, 1994, vol. 8.

[19] J. Leskovec and E. Horvitz, "Planetary-scale views on a large instant-messaging network," in *Proceedings of the 17th international conference on World Wide Web*. ACM, 2008, pp. 915–924.

[20] J. Donath and D. Boyd, "Public displays of connection," *bt technology Journal*, vol. 22, no. 4, pp. 71–82, 2004.

[21] K. Goodhope, J. Koshy, J. Kreps, N. Narkhede, R. Park, J. Rao, and V. Y. Ye, "Building linkedin's real-time activity data pipeline." *IEEE Data Eng. Bull.*, vol. 35, no. 2, pp. 33–45, 2012.

[22] K. M. M. Thein, "Apache kafka: Next generation distributed messaging system," *International Journal of Scientific Engineering and Technology Research*, vol. 3, no. 47, pp. 9478–9483, 2014.

[23] J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: A distributed messaging system for log processing," NetDB, 2011.

[24] A. Team, "Titan documentaion." [Online]. Available: http://s3.thinkaurelius.com/docs/titan/1.0.0/

[25] S. Schlicht, "Scale-out evaluation of news feed retrieval algorithms on neo4j and titan clusters," 2015.

[26] I. Tanase, Y. Xia, L. Nai, Y. Liu, W. Tan, J. Crawford, and C.-Y. Lin, "A highly efficient runtime and graph library for large scale graph analytics," in *Proceedings of Workshop on GRAph Data management Experiences and Systems*. ACM, 2014, pp. 1–6.

[27] M. A. Rodriguez, "The gremlin graph traversal machine and language (invited talk)," in *Proceedings of the 15th Symposium on Database Programming Languages*. ACM, 2015, pp. 1–10.

[28] S. Mallette, "Gremlin groovy." [Online]. Available: https://github.com/tinkerpop/gremlin/wiki/Using-Gremlin-through-Groovy

[29] M. A. Rodriguez, "Gremlin java." [Online]. Available: https://github.com/tinkerpop/gremlin/wiki/Using-Gremlin-through-Java

[30] M. Pollmeier, "Gremlin scala." [Online]. Available: https://github.com/mpollmeier/gremlin-scala

[31] S. Jouili and V. Vansteenberghe, "An empirical comparison of graph databases," in *Social Computing (SocialCom), 2013 International Conference on.* IEEE, 2013, pp. 708–715.

[32] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[33] N. Hayashibara, X. Defago, R. Yared, and T. Katayama, "The φ accrual failure detector," in *Reliable Distributed Systems, 2004. Proceedings of the 23rd IEEE International Symposium on.* IEEE, 2004, pp. 66–78.

[34] O. Kononenko, O. Baysal, R. Holmes, and M. W. Godfrey, "Mining modern repositories with elasticsearch," in *Proceedings of the 11th Working Conference on Mining Software Repositories.* ACM, 2014, pp. 328–331.

[35] M. Odersky, P. Altherr, V. Cremet, B. Emir, S. Maneth, S. Micheloud, N. Mihaylov, M. Schinz, E. Stenman, and M. Zenger, "An overview of the scala programming language," cole polytechnique fdrale de Lausanne, Tech. Rep., 2004.

[36] M. Odersky, "Scala: A scalable language"." [Online]. Available: http://www.scala-lang.org/what-is-scala.html

[37] M. N. La Polla, "Social media analytics and open source intelligence: the role of social media in intelligence activities," Ph.D. dissertation, University of Pisa, June 2014.

[38] E. Cambria, P. Chandra, A. Sharma, and A. Hussain, "Do not feel the trolls," *ISWC, Shanghai*, 2010.

[39] K. Freberg, K. Graham, K. McGaughey, and L. A. Freberg, "Who are the social media influencers? a study of public perceptions of personality," *Public Relations Review*, vol. 37, no. 1, pp. 90–92, 2011.

[40] M. E. Newman, "Assortative mixing in networks," *Physical review letters*, vol. 89, no. 20, p. 208701, 2002.

[41] S. P. Borgatti, "Centrality and network flow," *Social networks*, vol. 27, no. 1, pp. 55–71, 2005.

[42] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[43] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-worldnetworks," *nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[44] P. ERDdS and A. R&WI, "On random graphs i," *Publ. Math. Debrecen*, vol. 6, pp. 290–297, 1959.

[45] P. Erd6s and A. Rényi, "On the evolution of random graphs," *Publ. Math. Inst. Hungar. Acad. Sci*, vol. 5, pp. 17–61, 1960.

[46] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of modern physics*, vol. 74, no. 1, p. 47, 2002.

[47] N. developer team, "Networkx documetation." [Online]. Available: https://networkx.github.io/