Supporting End-to-End Social Media Data Analysis with the IndexedHBase Platform

Xiaoming Gao
School of Informatics and Computing,
Indiana University
201H Lindley Hall, 150 S. Woodlawn Ave.
Bloomington, IN 47405
1-812-272-6515
gao4@indiana.edu

Judy Qiu
School of Informatics and Computing,
Indiana University
201D Lindley Hall, 150 S. Woodlawn Ave.,
Bloomington, IN 47405
1- 812-855-4856
xqiu@indiana.edu

ABSTRACT
As data intensive applications evolve, many research projects involving Big Data require efficient extraction and analysis of specific data subsets, rather than the whole dataset. Social media data analysis is one such example. While social media platforms such as Twitter provide tremendous data about all kinds of social activities, most research analyses focus on specific social events, such as presidential elections or protests. In order to support the requirements of such research use cases, the storage platform needs to provide not only a scalable solution for the overall large dataset, but also mechanisms for efficiently querying the target subsets and applying post-query analyses. This paper introduces IndexedHBase, a storage platform specially designed to support end-to-end analysis of social media data. IndexedHBase uses HBase as the storage substrate, and provides a customizable indexing framework to facilitate queries about data subsets related to different social events. Beyond the queries, IndexedHBase can be integrated with parallel processing runtimes such as Hadoop and Twister to support sophisticated analysis of the query results through user-defined MapReduce functions. We describe the architecture and components of IndexedHBase, and demonstrate its effectiveness and efficiency by reproducing the end-to-end analysis of a published research project about the 2010 US midterm elections. We then extend this to a data subset about the 2012 presidential election, which serves to demonstrate that IndexedHBase correctly generates results that match with independent evaluations. Furthermore, our parallel implementation for the most time-consuming analysis step can achieve a processing speed that is tens to hundreds of times faster compared with a baseline sequential implementation in R for a distributed setup.

Categories and Subject Descriptors
H.2 [**Database Management**]: Systems.

General Terms
Algorithms, Performance, Design, Reliability, Experimentation.

Keywords
Social Media Data Analysis, IndexedHBase, Customizable Indexing Framework, Parallel Analysis.

1. INTRODUCTION
Motivated by the widespread adoption of social media platforms such as Twitter [31], investigating social activities through analysis of large scale social media datasets has been a popular research topic in recent years. Compared with previous data intensive computing problems, social media data analysis demonstrates a special characteristic: while the size of the whole social media dataset is huge, most analyses only focus on data subsets related to specific social events, or special aspects of social activities, such as congressional elections [8, 11], protest events [6, 7], and social link creation [33]. For such research scenarios, limiting analysis computation to the exact scope of the target subsets is important both in terms of efficiency and better resource utilization, especially in multi-task computing environments. Therefore, mechanisms for quickly locating the relevant data subsets are needed on the data storage and analysis platforms.

Another important feature about social data analysis is that the analysis workflow normally consists of multiple stages, and each stage may apply a diversity of algorithms to process the target data subsets, as illustrated in Figure 1. Implementations of certain algorithms may demonstrate different processing patterns. Therefore, to achieve efficient execution of the whole workflow, the analysis platform must adapt to different processing frameworks to complete various steps from these stages.

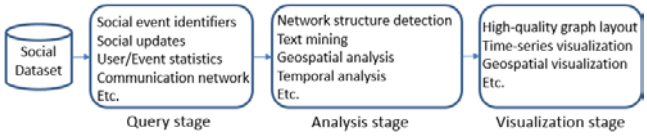

Figure 1. Stages in a social media data analysis process.

As a partial solution to these challenges, existing systems such as Eagle-Eyed-Elephant (E3) [15] and Hadoop++ [12] support efficient selection, aggregation, and join queries by building various indices over datasets stored in Hadoop Distributed File System (HDFS) [27]. However, queries supported by these systems cannot cover an end-to-end solution for the abovementioned scenarios, which may involve sophisticated mining and visualization processes over the query results. Moreover, the storage solution of HDFS does not support efficient random access of social updates (e.g. Twitter tweets), which is a basic requirement in many social data analysis projects.

In pursuit of a more complete solution, we introduce IndexedHBase [17], a storage platform that is specially designed for carrying out end-to-end social media data analysis. IndexedHBase has been used to serve the Truthy [18] social data observatory, and our previous work in [17] has demonstrated its scalability and

efficiency in handling Truthy queries. This paper extends our existing work and makes the following contributions:

(1) An extended architecture of IndexedHBase, which not only encapsulates efficient indexing and querying mechanisms, but can also be integrated with various parallel processing frameworks such as Hadoop [3] and Twister [14] to support sophisticated analysis of the query results through user-defined MapReduce functions.

(2) We provide implementations of two parallel algorithms that are generally useful for many social media data analysis processes. The first one is for mining related information about specific social activities, and involves processing of both original and index data. The second deals with visualization of query results that can be represented by a graph structure.

(3) A demonstration is given of the effectiveness and efficiency of IndexedHBase by reproducing the end-to-end analysis process from a published research project about political polarization [9], and further extending it to another data subset about the 2012 US presidential election. We validate our solution by comparing it with the published results, and investigate the scalability of our parallel program compared with a baseline implementation in R [29].

The rest of this paper is organized as follows. Section 2 describes the architecture and components of IndexedHBase, as well as its application in Truthy. Section 3 presents our implementations on IndexedHBase for reproducing an end-to-end analysis process about political polarization. Section 4 discusses related work. Section 5 concludes and gives prospects for our future work.

2. INDEXEDHBASE AND TRUTHY

2.1 Architecture of IndexedHBase

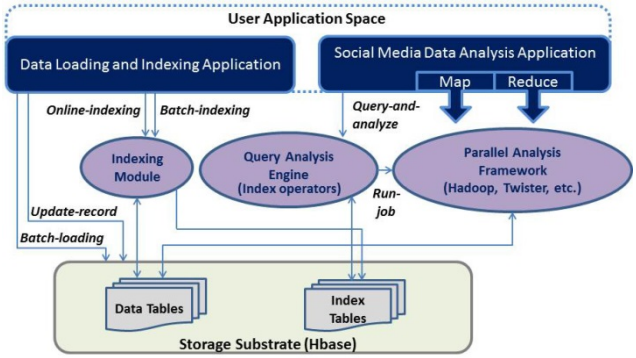

Figure 2. IndexedHBase Architecture.

Figure 2 shows the architecture of IndexedHBase. IndexedHBase employs the HBase [4] system as the storage substrate for both original social media data and generated index data. Leveraging a distributed architecture, HBase can provide reliable storage for TB- or even PB-level datasets. In addition it supports efficient random access as well as parallel scanning of Table data, which is a desirable feature for many social data analysis applications.

To quickly locate target data subsets, IndexedHBase has a flexible indexing module that can build customized index structures for data stored in its tables. Users can define these through an XML configuration file, as displayed in Figure 3. The file contains multiple "index-config" elements, each giving the mapping information from one source (data) table to one index table. Each index table implements one user-defined index structure, using its **row keys** to store the keys of the index, and its row content to store the corresponding entries. Within an "index-config" element, users can define which column of the source table will be indexed (in an HBase table, the combination of one **column family** and one **qualifier** specifies one column). The default content of each row contains only the **row keys** of the source table, but more information from the source table can be included to handle multi-dimensional queries. To construct more complicated index structures, a user-defined indexer class can also be specified.

Given any configuration file, the indexing module provides two mechanisms for building indices: *online-indexing* that indexes records of the data tables on the fly as they are dynamically inserted, and *batch-indexing* that builds index tables for pre-loaded data tables with Hadoop MapReduce jobs.

```
<?xml version="1.0" encoding="UTF-8"?>
<configurations>
    <index-config>
        <source-table>tweetTable</source-table>
        <source-column-family>details</source-column-family>
        <source-qualifier>text</source-qualifier>
        <source-value-type>full-text</source-value-type>
        <index-table>textIndexTable</index-table>
        <index-column-family>tweets</index-column-family>
        <index-qualifier>{source}.{rowkey}</index-qualifier>
        <index-timestamp>{source}.details.createdAt</index-timestamp>
        <index-value>{null}</index-value>
    </index-config>
    <index-config>
        <source-table>tweetTable</source-table>
        <index-table>memeIndexTable</index-table>
        <indexer-class>iu.pti.hbaseapp.truthy.tweetMemeIndexer</indexer-class>
    </index-config>
</configurations>
```

Figure 3. An example indexing configuration file.

Once the tables are built, they can be used to facilitate queries about data subsets through index operators. IndexedHBase automatically generates one default operator for each index table, which can find target subsets by directly matching information stored in the index table with a given set of parameters. Users can also implement their own operators to handle specially customized index structures. To complete analyses of queried social data subsets, user applications invoke the *query-and-analyze* interface of the query-analysis engine with three parameters: a query, a pair of user-defined map and reduce functions, and the type of parallel analysis platform to use. The query-analysis engine will first evaluate the queries by using combinations of the index operators, and then invoke the *run-job* interface of the corresponding framework to analyze the query results with the given map and reduce functions. The results will be split across multiple map tasks, which carry out the analysis computation on all splits in parallel. Currently two parallel analysis frameworks are supported: Hadoop [3] for simple one-pass MapReduce jobs, and Twister [14] for iterative MapReduce jobs.

2.2 IndexedHBase for Truthy

IndexedHBase is used to support Truthy [18], a public social media observatory that analyzes and visualizes information diffusion on Twitter. Truthy collects data through the Twitter streaming API [30], which provides a stream that includes a sample of public tweets from Twitter. Currently, the total size of historical data collected since August 2010 is approximately 10 Terabytes in compressed format, and the data rate coming out of the dynamic stream is in the range of 45-50 million tweets per day, leading to a growth of approximately 20GB in the total data size. Each tweet comes in the form of a structured JSON string containing information about both the tweet and the user who posted it. Furthermore, if the tweet is a retweet, the original content is also included in a "retweeted_status" field. Figure 12 in the appendix illustrates the structure of an example tweet.

To support efficient loading and access of this dataset, we design the two data tables as shown in Figure 4. The tweet table uses tweet IDs as **row keys**, and each row contains multiple columns under a single **column family** called "details". Truthy uses the concept of "meme" to represent a set of related posts corresponding to a specific discussion topic, communication channel, or information source shared by users. Memes can be identified through elements contained in the texts of tweets, such as *keywords*, *hashtags* (e.g., #euro2012), *user-mentions* (e.g., @youtube), and *URLs*. Correspondingly, the tweet table contains a "text" column for the text content, and a "memes" column to store the elements. The user table uses a concatenation of user ID and tweet ID as the **row key** to track all the changes of each user's metadata associated with each tweet he/she has posted.

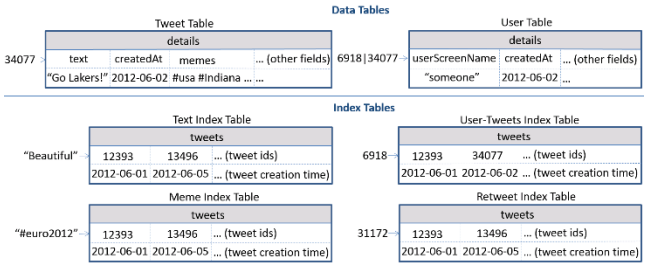

Figure 4. HBase tables for Truthy.

The dataset of Truthy has been used in research projects covering a broad spectrum of social activities, including political polarization [9], congressional elections [8, 11], protest events [6, 7], and the spread of misinformation [23, 24]. Based on these projects, Truthy proposes a set of queries that are generally useful for most of them. These queries can be divided into two categories: *basic queries* and *advanced queries*.

Basic queries involve search of tweet subsets according to certain criteria. For example, given two parameters *memes* and *time-window*, where *memes* is a list of hashtags, user-mentions, or URLs, and *time-window* is a pair of time points, the query ***get-tweets-with-meme***(*memes*, *time-window*) tries to find the IDs of all tweets containing any elements in *memes* and created during the given *time-window*. Other basic queries include ***get-tweets-with-text*** (*keywords*, *time-window*), ***get-tweets-with-user*** (*user-id*, *time-window*), and ***get-retweets*** (*tweet-id*, *time-window*). To achieve efficient evaluation of these queries, we design the index tables in Figure 4. Each table uses values from the indexed columns of the tweet table as **row keys**, related tweet IDs as column names (**qualifiers**), and creation time of the corresponding tweets as **timestamps**. All basic queries can be evaluated by using the default operators of these index tables. So to evaluate ***get-tweets-with-meme***(*memes*, *time-window*), we can simply use the operator for the meme index table to select the tweet IDs associated with qualified creation time.

Advanced queries apply further processing of basic queries and return the final results. These include ***timestamp-count***, ***user-post-count***, ***meme-post-count***, ***meme-cooccurrence-count***, ***get-retweet-edges***, and ***get-mention-edges***. All queries take two parameters, *memes* and *time-window*, and require processing of the tweets returned by ***get-tweets-with-meme***. Each query can actually be implemented as one *query-and-analyze* process on IndexedHBase. For instance, ***get-retweet-edges*** uses a Hadoop MapReduce job to process the tweet IDs found by ***get-tweets-with-meme***. A map task takes a subset of tweet IDs and checks the content of each corresponding tweet. If it is a retweet, the map task will output a <*key*, *value*> pair, where *key* is a retweet edge containing a source user ID and a retweeted user ID, and *value* is 1. The reducer tasks simply collect the output of all mappers and generate retweet edges associated with their weights. Note that efficient random access to the tweet table is critical for this process, because the related tweet IDs are not necessarily consecutively stored in the table. Details about implementations of the other queries are explained in [17].

Most social media data analysis processes on Truthy start with execution of one or multiple such queries, and involve extended analysis and visualization of the query results.

3. REPRODUCING END-TO-END ANALYSIS ON INDEXEDHBASE

This section details an application of IndexedHBase by reproducing the end-to-end analysis process presented in a published research project [9] using the dataset of Truthy. The project investigated how social media shape the networked public sphere and facilitate communication between communities with different political orientations. More than 250,000 politically relevant tweets were extracted from the Truthy dataset during the six weeks leading up to the 2010 U.S. congressional midterm elections. Then the characteristics of the *retweet network* and *mention network* generated from these tweets were examined. The results showed that the retweet network exhibited a highly modular structure, segregating users into two homogenous communities corresponding to the political left and right. In contrast, the mention network did not exhibit such political segregation.

We will first try to reproduce the analysis and results in [9] on IndexedHBase using the same dataset from 2010, and then extend the same analysis process to another dataset collected by Truthy during the six weeks before the 2012 U.S. presidential election to verify if a similar pattern in the social communication networks can be observed. Our explanation in this paper focuses on analysis of the retweet network, and implementations for the mention network are similar.

3.1 Analysis Workflow

Figure 5 illustrates the major steps of the analysis process in [9].

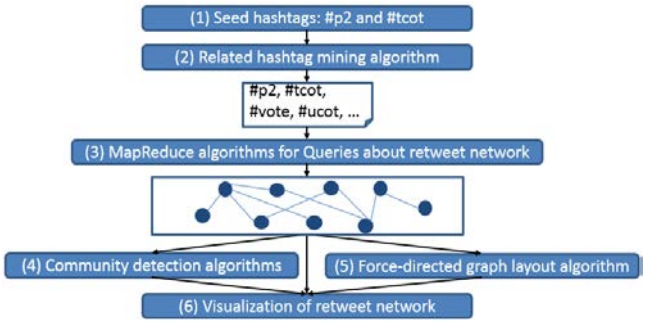

Figure 5. End-to-end analysis workflow in [9].

As an important feature of Twitter, *hashtags* are widely used to annotate social updates as indications of relevant topics and intended audience. Therefore, the first two steps in the workflow try to find a set of political hashtags that can be used to identify politically related tweets from all those collected during the selected six-week time window. In **Step (1)**, two of the most popular political hashtags, #p2 ("Progressives 2.0") and #tcot ("Top Conservatives on Twitter") are manually selected as seed hashtags. **Step (2)** tries to extend this initial set with other related hashtags by calculating the Jaccard coefficient between a seed hashtag and others that have co-occurred with it in at least one tweet. For a set of tweets *S* containing a seed hashtag, and another

set *T* containing a different hashtag, the Jaccard coefficient between *S* and *T* is defined as:

$$\sigma(S,T) = \frac{|S \cap T|}{|S \cup T|}. \quad (1)$$

When this coefficient is large enough, the two hashtags are recognized as related. A threshold of 0.005 was used in [9] to identify the related hashtags for both seeds.

Step (3) executes the ***get-retweet-edges*** query, using all hashtags found in Step (2) as the *memes* parameter and the six-week time window as the *time-window* parameter. It does this to get the retweet network among users from both political orientations. The retweet edges compose a graph structure, with vertices representing users and edges representing 'retweet' relationships that happened during the time window.

Step (4) uses a combination of two algorithms, leading eigenvector modularity maximization [21] and label propagation [22], to detect communities on the retweet network. Here a "community" is defined as a set of vertices on a graph that are densely inter-connected and sparsely connected to the other parts of the graph. After this step, vertices from different communities are labeled with different colors for visualization in Step (6).

In order to achieve a high-quality visualization of segregated communities in the retweet network, **Step (5)** uses the "Fruchterman-Reingold" force-directed layout algorithm [16] to generate a desirable layout of the retweet network. This algorithm goes through multiple iterations computing vertices' positions to achieve a layout where inter-connected nodes are 'pulled' towards each other and disconnected nodes are 'pushed' apart.

Step (6) makes a final plot of the retweet network from Step (3) using the color labels computed in Step (4) and layout information generated in Step (5).

3.2 Reproducing Results for 2010

The following represents our implementation for reproducing the analysis workflow in Figure 5, and validates our solution by comparing against the ones originally reported in [9]. Our experiments are carried out on 35 nodes of the Alamo HPC cluster on FutureGrid [32]. The hardware configuration is given in Table 1. Each node is installed with CentOS 5.9 and Java 1.7.0_40. We use Hadoop 1.0.4, HBase 0.94.2, Twister 0.9 (together with ActiveMQ 5.4.2), and R 2.10.1 in our experiments. Among the 35 nodes, one is used to host the Hadoop jobtracker and HDFS namenode, another hosts the HBase master, and a third hosts Zookeeper and Active MQ broker. The other 32 nodes host HDFS datanodes, Hadoop tasktrackers, HBase region servers, and Twister daemons.

Table 1. Per-node configuration on the Alamo Cluster

CPU	RAM	Hard Disk	Network
8 * 2.66GHz (Intel Xeon X5550)	12GB	500GB	40Gb InfiniBand

As explained in Section 3.1, **Step (1)** is fixed to manual choice of #p2 and #tcot. We implement **Step (2)** as two *query-and-analyze* processes on IndexedHBase, one for finding related hashtags for #p2, and one for #tcot. Both processes set the query to ***get-tweets-with-meme***, and apply analysis over the query results with the map and reduce functions as given in Figure 6.

After getting the tweet IDs for the seed hashtag, the query-analysis engine will automatically split them in to multiple partitions, each containing a fixed number (which is set to 30,000 by default) of tweet IDs. A Hadoop MapReduce job is then scheduled to process all the partitions in parallel with the functions given in Figure 6. Each mapper processes one partition, and for every tweet ID in that partition, the mapper will access the corresponding row in the tweet table and get the value of the "memes" column, which contains all hashtags, user-mentions, and URLs from the corresponding tweet. Then the mapper will output all the hashtags that are different from the seed hashtag as intermediate results. After the shuffling phase, each reducer will receive a list of unique hashtags that have co-occurred with the seed hashtag. For each hashtag in the list, the reducer uses the ***get-tweets-with-meme*** operator to access the meme index table and find the related tweet IDs. Then the Jaccard coefficient between this hashtag and the seed hashtag is calculated according to formula (1); if the value reaches the given threshold (0.005), this hashtag will be output as a final result. Overall, it takes 109.3 seconds to find related hashtags for #p2, which involves analysis of the content of 109,312 tweets with 4 map tasks. The same process for #tcot spends 128.1 seconds in analyzing 189,840 tweets with 8 map tasks. Merging the results for both seed hashtags, we found the same 66 related hashtags as [9].

```
1  seed = given hashtag;     /* #p2 or #tcot */
2  tw = given time window; /* six weeks before 2010 congressional elections */
3  seedTids = set of IDs of tweets containing seed;
4  th = given threshold for Jaccard Coefficient; /* 0.005 */
5
6  function map(key, value) {
7   tweetID = value;
8   tweet = access the tweet table with tweetID as row key;
9   memes = tweet.getColumnValue("details", "memes");
10  for each meme in memes {
11   if meme is a hashtag and meme != seed {
12    output(meme, NULL);
13   }
14  }
15 }
16
17 function reduce(key, List<value>) {
18  hashtag = key;
19  htTids = access the index table with get-tweets-with-meme(hashtag, tw);
20  coeff = |seetTids ∩ htTids| / |seedTids ∪ htUids|;
21  if coeff >= th {
22   output(hashtag, NULL);
23  }
24 }
```

Figure 6. MapReduce algorithm for mining related hashtags.

Our implementation for Step (2) demonstrates that index data is not only useful for query evaluation, but also valuable for analysis purposes such as mining of related hashtags. The algorithm in Figure 6 is generally useful for all social data analysis projects that need to find a set of related hashtags based on seed hashtags.

Step (3) is directly completed with the ***get-retweet-edges*** query. This step takes 93.3 seconds, and returns the same retweet network as in [9], which contains 23,766 non-isolated nodes.

Step (4), (5), and (6) can be completed by using the igraph [28] library of R, which provides a baseline benchmark with sequential implementation. Table 2 lists the execution time of these three steps with R on a single node. It can be observed that Step (5) is significantly more time consuming than the other two steps, and may potentially become a bottleneck of the analysis workflow as we apply it to larger-scale datasets. Therefore, we provide a parallel implementation of the "Fruchterman-Reingold" layout algorithm [16] to speed up the whole workflow. Since the algorithm involves iterative computation, our implementation is based on Twister, which provides better performance than Hadoop for iterative

algorithms [14]. The parallel "Fruchterman-Reingold" algorithm (MRFR) is given in Figure 7.

Table 2. Execution time (seconds) for Step (4) - (6)

(4) Community Detection	(5) Graph Layout (500 iterations)	(6) Visualization
3.4	4508.3	1.6

The idea of the "Fruchterman-Reingold" algorithm is to compute the layout of a graph by simulating the behavior of a physical system where vertices of the graph are taken as atomic particles, and edges are taken as springs. A repulsive force exists between each pair of atomic particles, which tends to push them away from each other. An attractive force exists on each spring, which tends to pull the vertices at the two ends closer to each other. Both forces are defined as functions of distances between vertices. Therefore, starting from an initial state of random layout, in each iteration, disconnected vertices are pushed further apart, and vertices connected with edges are pulled closer together. Over multiple iterations, the whole system eventually evolves to a 'low-energy' state. Besides the forces, a "temperature" parameter is used to limit the maximum displacement of vertices in each iteration. The temperature eventually 'cools' down as iterations go.

The whole Step (5) is also implemented as a *query-and-analyze* process. To facilitate this step, we modified ***get-retweet-edges*** to get ***get-retweet-adjmtx***, a new query that generates the **adjacency matrix** of the retweet network instead of only the edges. This query outputs a list of lines, and each line is in the form of '<vertex ID> <neighbor vertex ID> <neighbor vertex ID> …' i.e., a vertex ID followed by a list of IDs of other vertices that are connected with this vertex by edges. After executing ***get-retweet-adjmtx***, the query-analysis engine partitions the adjacency matrix into multiple sub-graphs, each containing a subset of vertices associated with their neighbors. Then an iterative MapReduce job is scheduled on Twister to process these sub-graphs in parallel, using the functions given in Figure 7. The whole job works as follows: during job initialization time, an initial random layout of the whole graph is broadcasted to all the mappers. Each mapper reads in a sub-graph during task initialization time, then saves it in memory for usage across all iterations. Within every iteration, each mapper receives the global layout of the whole graph from the last iteration through its input <*key*, *value*> pair. Then for each vertex in the sub-graph, the mapper first calculates its displacement based on the repulsive forces it receives from every other vertex, and again based on the attractive forces it receives from its neighbors, and finally decides its total displacement by taking the temperature into consideration. Then a new layout of the sub-graph is generated based on the displacements, and output as an intermediate result from the mapper. The reducer collects the output from all mappers to generate the global layout. If the maximum number of iterations is reached, the reducer will output the global layout as the final result. Otherwise, the global layout is broadcasted to all mappers for the next iteration.

Figure 8 illustrates the per-iteration execution time and speed-ups of MRFR under different levels of parallelism. It is obvious that MRFR can effectively speed up the graph layout step. Specifically, with 64 mappers on 8 nodes, MRFR runs 15 times faster than the sequential implementation in R, completing 500 iterations within 300 seconds. However, MRFR does not achieve very good scalability for the 2010 retweet network, mainly because the amount of computation required in mappers is not large enough compared to the scheduling and communication overhead. For example, in case of 64 mappers, the slowest mapper finishes in 250ms, while the total overhead stays consistently at about 350ms across different numbers of mappers. Figure 9 shows the final visualization of the retweet network using the layout generated by MRFR. The layout is almost the same as the plot in [9], with only a slight difference caused by a different initial random layout. As identified in [9], the red cluster is made of 93% right leaning (conservative) users, and the blue cluster is made of 80% left leaning (progressive) users. Since we generate the same result as [9] in each step of the analysis workflow, our solution on IndexedHBase is validated.

```
1  V = total number of vertices in the whole graph;
2  area = V * V;
3  maxDelta = V;
4  k = sqrt(area/V);
5  nIter = max number of iterations as given;  /* 500 */
6  curIter = 0;                                /* current iteration count */
7  function fa(dist) { return dist²/k; }       /* attractive force */
8  function fr(dist) { return k²/dist; }       /* repulsive force */
9
10 function map(key, value) {
11   sg = sub-graph for this mapper;           /* read-in during initialization */
12   globalLo = value;                         /* global layout from last iteration */
13   t = maxDelta * pow((nIter – curIter)/niter, 1.5);  /* temperature */
14   sgLo = ∅;                                 /* sub-graph layout */
15   for each v in sg {
16     v.disp = 0;
17     for each u ≠ v in globalLo {            /* displacement by repulsive force */
18       Δ = v.pos – u.pos;
19       v.disp = v.disp + (Δ/|Δ|) * fr(|Δ|);
20     }
21     for each n in sg.getNeighbors(v) {      /* displacement by attractive force */
22       nPos = globalLo.getPositionOf(n);
23       Δ = v.pos – nPos;
24       v.disp = v.disp - (Δ/|Δ|) * fa(|Δ|);
25     }
26     /* limit displacement by temperature */
27     if |v.disp| > t { v.disp = v.disp * t / |v.disp| }
28     v.pos = v.pos + v.disp;
29     sgLo = sgLo ∪ v.pos;
30   }
31   output("dummy-key", sgLo);
32 }
33
34 function reduce(key, List<value>) {
35   globalLo = ∅;                             /* global layout for this iteration) */
36   for each value in List<value> { globalLo = globalLo ∪ value; }
37   curIter++;
38   if curIter >= nIter {
39     outputAndExit(globalLo, NULL);
40   } else {
41     broadcast(globalLo);
42   }
43 }
```

Figure 7. Parallel Fruchterman-Reingold algorithm using iterative MapReduce.

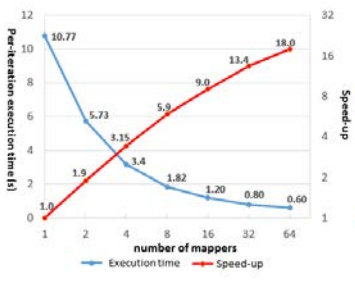

Figure 8. Per-iteration execution time for MRFR.

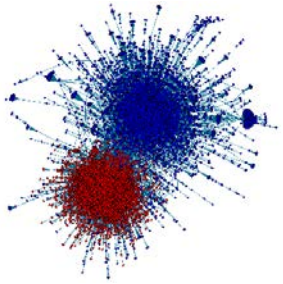

Figure 9. Final plot of the retweet network.

3.3 Extending Analysis to Data in 2012

Here we extend the analysis workflow in Figure 5 to a later dataset collected during the six weeks (09/24/2012 to 11/06/2012) before the 2012 U.S. presidential election, and verify if the corresponding

retweet network demonstrates a similar polarized pattern. The average data size for each day in 2012 is about 6 times larger than 2010.

Step (1) still starts from #p2 and #tcot. **Step (2)** spends 142 seconds in mining related hashtags for #p2, and 191 seconds for #tcot. The number of tweets analyzed is 160,934 and 364,825 respectively. In total, 66 related hashtags are found (see Table 3 in the appendix). In **Step (3)**, 80 mappers need 150 seconds to analyze 2,360,361 politically related tweets, and the result is a retweet network that is 20 times larger, with 477,111 vertices and 665,599 edges.

Step (4) requires 2402 seconds on R to complete community detection for this large network. In **Step (5)**, it takes as long as 6044 seconds to finish only one iteration of the Fruchterman-Reingold algorithm on R. This demonstrates that due to the fast growth of data volume, sequential algorithms quickly become infeasible for social data analysis scenarios. In order to address this challenge, we use more mappers in MRFR to complete Step (5). Figure 10 illustrates the per-iteration execution time and speed-ups of MRFR for the 2012 retweet network. The near-linear scalability clearly demonstrates that MRFR is especially good at handling large networks. In particular, using 256 mappers on 32 nodes, MRFR can finish one iteration 355 times faster than the sequential implementation in R. **Step (6)** runs for 32 seconds on R, and Figure 11 shows the final plot of the two largest communities of the retweet network. On the one hand, we can still observe a clearly segregated political structure in the 2012 network; on the other hand, the two sides also seem to demonstrate a 'merging' trend by having more edges reaching out to each other.

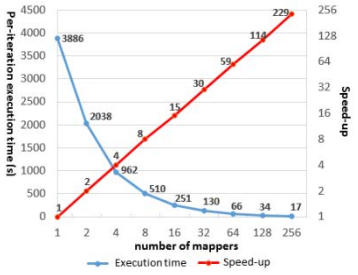

Figure 10. Per-iteration execution time of MRFR (2012).

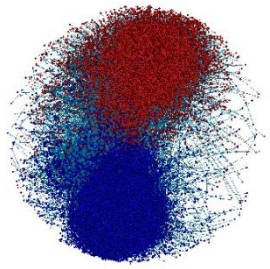

Figure 11. Final plot of the retweet network (2012).

4. RELATED WORK

For details about the data loading, indexing, and query evaluation strategies of IndexedHBase, please refer to [17]. For a list of other social media data analysis projects that can be supported by IndexedHBase, please refer to [6, 7, 8, 11, 23, 24, 33].

DataStax (Cassandra) [10] and Riak [25] are two other systems that also use distributed NoSQL databases for data storage and in addition support queries about data subsets with text and secondary indices. However, since Cassandra does not support range scans very well, it is not suitable for several important use cases in social data analysis, e.g. range queries about memes in the form of '#occupy*'. Moreover, the indexing mechanisms in these systems are designed mainly for search purposes, and thus are neither customizable nor flexible enough for efficient evaluation of the temporal queries and analyses in Truthy. As demonstrated in [17], the lightweight MapReduce framework of Riak cannot handle the result size of the queries in Truthy.

Hadoop++ [12], HAIL [13], and Eagle-Eyed Elephant [15] are systems that try to extend the Hadoop [3] system with various indexing mechanisms to facilitate MapReduce queries. However, the queries targeted by these systems do not cover sophisticated analytics that may involve iterative computation. Besides, they all schedule MapReduce tasks based on data blocks or splits (or at least 'relevant' splits) stored on HDFS, and tasks may have to scan irrelevant data during query evaluation. In contrast, by leveraging HBase for efficient random access of data records, IndexedHBase can dynamically adjust the number of MapReduce tasks in a job according to the exact number of records in the target data subsets, and the tasks only need to access relevant data records to produce the final result.

HadoopDB [1] provides a hybrid solution that can utilize the indexing techniques provided by relational databases to achieve efficient query evaluation. However, HadoopDB applies deep changes to the Hadoop framework, and forces the use of relational databases in a parallel architecture, which is difficult to configure and maintain. The SQL queries supported by HadoopDB also do not cover sophisticated iterative analysis algorithms.

By using Spark [35] as the execution engine and applying various optimizations to its in-memory processing model, Shark [34] is able to support both efficient SQL queries and sophisticated iterative analytics at a large scale. Compared with Shark, IndexedHBase supports efficient fine-grained data operations, putting an emphasis on building suitable index structures to facilitate location of target data subsets. IndexedHBase can be integrated with Shark, and help further improve the performance of analysis jobs by only loading relevant data records as RDDs in Spark. The columnar storage of table data used by both Shark and Dremel [19] are inspiring to IndexedHBase in terms of more efficient query evaluation and iterative analyses.

To the best of our knowledge, our algorithm in Figure 7 is the first iterative MapReduce implementation for the Fruchterman-Reingold layout algorithm. There are previous efforts on parallelizing this algorithm with MPI [20] and GPUs [26], and we may consider extending our solution with the usage of GPUs on each node to handle larger-scale problems.

5. CONCLUSIONS AND FUTURE WORK

In the end we gained three important lessons from our experience with IndexedHBase. First, flexible indexing mechanisms and efficient random access to single data records are two critical factors for fast location of target data subsets. Second, index data is not only useful for query evaluation, but also valuable for analysis and mining purposes. Finally, social data analysis workflows normally consist of multiple tasks that are suitable for different processing patterns. As such, dynamically adopting different frameworks to handle different tasks is crucial to achieve efficient processing of the whole workflow.

There are two major directions that we consider worthy of future work. First, we will integrate IndexedHBase with other parallel processing frameworks like Giraph [2] to handle more variations in the computation pattern of social data analysis tasks. Second, we will try to extend IndexedHBase with a high level language such as Pig [5] to facilitate composition of complex workflows.

6. ACKNOWLEDGMENTS

We would like to thank the Truthy group of IU School of Informatics and Computing for their collaboration and help on our questions about social media data analysis. Thanks to Prof. Geoffrey Fox for his advice on development of IndexedHBase. Thanks also to the FutureGrid team for their help on our testing environment setup.

7. REFERENCES

[1] Abouzeid, A., et al. 2009. HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads. *Proc. VLDB Endow.* 2, 1 (August 2009), 922-933.

[2] Apache Giraph. http://giraph.apache.org/.

[3] Apache Hadoop. http://hadoop.apache.org/.

[4] Apache HBase. http://hbase.apache.org/.

[5] Apache Pig. http://pig.apache.org/.

[6] Conover, M., et al. 2013. The Geospatial Characteristics of a Social Movement Communication Network. *PLoS ONE*, 8(3): e55957.

[7] Conover, M., et al. 2013. The Digital Evolution of Occupy Wall Street. *PloS ONE*, 8(5), e64679.

[8] Conover, M., et al. 2012. Partisan asymmetries in online political activity. *EPJ Data Science*, 1, 6 (2012).

[9] Conover, M., et al. 2011. Political Polarization on Twitter. In *Proceedings of the 5th International AAAI Conference on Weblogs and Social Media, (ICWSM 2011).*

[10] DataStax. http://www.datastax.com/.

[11] DiGrazia, J., et al. 2013. More Tweets, More Votes: Social Media as a Quantitative Indicator of Political Behavior. Available at SSRN: http://dx.doi.org/10.2139/ssrn.2235423

[12] Dittrich, J., et al. 2010. Hadoop++: making a yellow elephant run like a cheetah (without it even noticing). *Proc. VLDB Endow.* 3, 1-2 (September 2010), 515-529.

[13] Dittrich, J., et al. 2012. Only aggressive elephants are fast elephants. *Proc. VLDB Endow.* 5, 11 (July 2012), 1591-1602.

[14] Ekanayake, J., et al. 2010. Twister: a runtime for iterative MapReduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC '10)*. ACM, New York, NY, USA, 810-818.

[15] Eltabakh, M., et al. 2013. Eagle-eyed elephant: split-oriented indexing in Hadoop. In *Proceedings of the 16th International Conference on Extending Database Technology (EDBT '13).* ACM, New York, NY, USA, 89-100.

[16] Fruchterman, T., Reingold, E. 1991. Graph drawing by force-directed placement. *Softw. Pract. Exper.* 21, 11 (November 1991), 1129-1164.

[17] Gao, X., et al. 2013. Supporting a Social Media Observatory with Customizable Index Structures - Architecture and Performance. Book chapter to appear in *Cloud Computing for Data Intensive Applications*, to be published by Springer Publisher, 2014.

[18] McKelvey, K., Menczer, F. 2013. Design and prototyping of a social media observatory. In *Proceedings of the 22nd international conference on World Wide Web companion (WWW '13 Companion).* International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 1351-1358.

[19] Melnik, S., et al. 2010. Dremel: interactive analysis of web-scale datasets. *Proc. VLDB Endow*. 3, 1-2 (September 2010), 330-339.

[20] Mueller, C., Gregor, D., Lumsdaine, A. 2006. Distributed force-directed graph layout and visualization. In *Proceedings of the 6th Eurographics conference on Parallel Graphics and Visualization (EG PGV'06).* Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 83-90.

[21] Newman, M. 2006. Finding community structure in networks using the eigenvectors of matrices. *Physical Review* E 74, 036104 (2006).

[22] Raghavan, U., Albert, R., Kumara, S. 2007. Near linear time algorithm to detect community structures in largescale networks. *Physical Review* E 76, 036106 (2007).

[23] Ratkiewicz, J., et al. 2011. Truthy: mapping the spread of astroturf in microblog streams. In *Proceedings of the 20th international conference companion on World wide web (WWW '11).* ACM, New York, NY, USA, 249-252.

[24] Ratkiewicz, J., et al. 2011. Detecting and Tracking Political Abuse in Social Media. In *Proceedings of 5th International AAAI Conference on Weblogs and Social Media (ICWSM 2011).*

[25] Riak. http://basho.com/riak/.

[26] Sharma, P., et al. 2011. Speeding up network layout and centrality measures for social computing goals. In *Proceedings of the 4th international conference on social computing, behavioral-cultural modeling and prediction (SBP'11).* Springer-Verlag, Berlin, Heidelberg, 244-251.

[27] Shvachko, K., et al. 2010. The Hadoop Distributed File System. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST) (MSST '10).* IEEE Computer Society, Washington, DC, USA, 1-10.

[28] The igraph library. http://igraph.sourceforge.net/.

[29] The R Project. http://www.r-project.org/.

[30] Twitter Streaming API. https://dev.twitter.com/docs/streaming-apis.

[31] Twitter. https://twitter.com/.

[32] Von Laszewski, G., et al. 2010. Design of the FutureGrid Experiment Management Framework. In *Proceedings of Gateway Computing Environments Workshop, (GCE 2010).*

[33] Weng, L., et al. 2013. The role of information diffusion in the evolution of social networks. In *Proceedings of 19th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, (KDD 2013).*

[34] Xin, R., et al. 2013. Shark: SQL and rich analytics at scale. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13).* ACM, New York, NY, USA, 13-24.

[35] Zaharia, M., et al. 2012. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI'12).* USENIX Association, Berkeley, CA, USA, 2-2.