International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 7, Number 1, pages 86–104, January 2017

Finding Key Persons on Social Media by Using $MapReduce$ Skyline

Asif Zaman
Graduate School of Engineering, Hiroshima University
Kagamiyama 1-7-1, Higashi-Hiroshima 739-8521, Japan
Email: d140094@hiroshima-u.ac.jp

Md. Anisuzzaman Siddique
Graduate School of Engineering, Hiroshima University
Kagamiyama 1-7-1, Higashi-Hiroshima 739-8521, Japan
Email: siddique@hiroshima-u.ac.jp, anis_cst@yahoo.com

Annisa
Graduate School of Engineering, Hiroshima University
Kagamiyama 1-7-1, Higashi-Hiroshima 739-8521, Japan
Email: d144809@hiroshima-u.ac.jp

and

Yasuhiko Morimoto
Graduate School of Engineering, Hiroshima University
Kagamiyama 1-7-1, Higashi-Hiroshima 739-8521, Japan
Email: morimoto@mis.hiroshima-u.ac.jp

Received: February 15, 2016
Revised: April 30, 2016
Revised: August 30, 2016
Revised: November 19, 2016
Accepted: November 25, 2016
Communicated by Hiroyuki Sato

Abstract

This study considers the problem of selecting a small number of important persons from social media. Skyline query has been utilized for selecting key persons. Based on certain criteria from social media, this query selects persons who are not dominated by any other. Owing to the complex structure of social media, selecting a key person is more complicated and its application is quite different from conventional skyline queries. We need to consider various metrics in the social media. In addition, social media contains massive data, and the data increase is huge. It is collection of online communication channels dedicated to community-based inputs, interactions, content sharing, and collaboration. We use *MapReduce* framework to speed up the computation and introduce parallelism in the processing. An extensive set of experiments shows that the analysis of social activities, social relationships, and socially shared contents helps finding a key person. The experimental results also confirm the efficiency and scalability of our algorithm on a synthetic dataset.

Keywords: Social networks, Key person, Skyline query, MapReduce.

1 Introduction

At every moment, enormous amounts of data are generated on social media. Such huge amounts of data can be used to interpret people and predict their behaviors more diligently. Social media or social network is a collection of online communication channels dedicated to community-based inputs, interactions, content sharing, collaboration, and etc. Examples of social media includes websites and apps dedicated to forums, micro-blogging, social networking, social bookmarking, and wikis.

Data mining research has successfully produced numerous methods, tools, and algorithms for handling large amounts of data to solve real-world problems. Therefore, data mining has become an integral part of many application domains including bioinformatics, data warehousing, business intelligence, predictive analytics, and decision support systems. The primary objectives of the data mining process are to effectively handle large-scale data, extract actionable patterns, and gain insightful knowledge. Nowadays, social media is widely used for various purposes. Vast amounts of user-generated data exist and it can be made available for various kinds of analysis. There is no doubt that to utilize such data in social media is the key to improve various business processes.

In this study, we consider the problem of selecting a small number of key persons from a social media database. As a model of social media, we selected the data from Facebook database because of its usefulness, reputation, and popularity.

Skyline queries, which select non-dominated objects, are known to be useful to select a small number of preferable objects from a database [6, 13, 20]. We use the idea of skyline query to solve the key person selection problem. However, selecting a key person from the Facebook database is more complicated as compared to a general skyline query, because it is different from general relational tables where there are attributes and their corresponding domain values. We must consider the different metrics in social media to handle large datasets. The metrics in social media may include the inter-personal relationship (e.g. friend, follower), user's group membership, their "comments," "comment feedback," "likes," picture and status "shares," "blocks," etc. In this work, we consider friends, followers, "like" events, comment feedback, and memberships for different groups to select the key person.

A symbolic Facebook database is illustrated in Figure 1. Let us assume that we want to select a small number of key persons from this symbolic dataset. We consider following criteria to select the key person:

1. Friend power − total number of friends that a person has in a social network
2. Followee strength − total number of followers
3. "Like" score − average "like" count
4. Comment support − based on positive and negative comment replies
5. Group score − sum of the group scores of all groups to which the user belongs.

We assume a key person has a large number of friends, followers, higher average "like" count, higher comment score; he/she also has the membership to important groups. User U *dominates* another user U' if all the five criteria of U are better than or equal to those of U''s and in at least one of the five criteria of U is better than that of U'.

In Figure 1, the first, second, and third columns represent user id (denoted as UID) of social media, friend list, and followee list (list that follows the UID), respectively. The fourth column represents the "like" records in the pattern of $\langle ID_{post}, UID_{liker} \rangle$, where ID_{post} is the unique ID of different *posts* or *status update* posted by different users in social media, and UID_{liker} is the user id of the person who has given a "like" to that post or status update. Facebook does not support any "dislike" event. Similarly, the fifth column represents the comment feedback form different users in the pattern of $\langle ID_{post}, UID_{replyer}, Comment_{fdbk} \rangle$, where ID_{post} is the unique id of different status updates posted in social media, $UID_{replyer}$ is the user id of the commentator who has posted a feedback comment to that status post, and the $Comment_{fdbk}$ is the text that has been replied by the $UID_{replyer}$. Note that $Comment_{fdbk}$ could have been a neutral, positive or negative feedback. For simplicity, in this work, we did not consider comments on photos. The sixth column represents the group membership of each user. We put 1 if UID is a member of that group, otherwise we leave the cell blank.

The first record of Figure 1 shows that user "A" has several friends (C, D, G, K,$\cdots$) in the friends list, in which we assume the friend number count as 34 and the followee count as 81. User "D" has given a

UID	Friends	Followee	Like Records	Comment Feedback	Groups				
					G1	G2	G3	G4	G5
A	C, D, G, K, …(34)	B, L, H, …(81)	(5, D),(7, D)...	(36, C, $Comment_1$), (26, D, $Comment_3$)…	1			1	1
B	D, H, ...(20)	A, F, W, …(73)	(79, E), (79,K)…	(27, C, $Comment_2$), (78, K, $Comment_8$)…	1	1	1		
C	A, E, …(55)	L, H, …(32)	(61, A),(65, E)…	(42, F,$Comment_{15}$)…			1		1
D	A, E, F, H, … (75)	B, C, …(40)	(33,B),(33,A)…	(31, B, $Comment_6$)…	1	1	1	1	
E	D, F, C, G, …(72)	A, B, …(96)	(101,L),(107,C)…	(11, L, $Comment_{17}$)…	1	1	1	1	1
F	E, D, …(94)	P, Q, …(56)	(201, D),(209, L)…	(20, D $Comment_{11}$)…		1	1		1
G	A, E, …(63)	M, N, O, …(63)	(301, P),(308,F)…	(6, P, $Comment_{12}$)…		1		1	1
H	B, D, …(62)	A, Q, M, …(71)	(307, J), (455, I)…	(37, C, $Comment_{19}$)…	1	1		1	
I	K, L, …(30)	O, P, R, …(23	(510, S), (544,U)…	(36, L, $Comment_{18}$)…	1				1
J	P, Q, R, …(46)	A, B, C, …(57)	(515, A), (515,C)…	(45, B, $Comment_7$)…	1	1			
…	…	…	…	…	…	…	…	…	…

Figure 1. Example of Facebook data

"like" to the 5^{th} and 7^{th} status update of user "A," meanwhile users "C" and "D" have replied as $Comment_1$ and $Comment_3$ to the 36^{th} and 26^{th} "status update" of user "A," respectively. In addition, user "A" has membership of groups $G1$, $G4$, and $G5$. In our example, there are five groups: Carrier support ($G1$), Sports ($G2$), Video club ($G3$), Photography ($G4$), and Tourist ($G5$). As mentioned earlier, if a person is a member of some particular group, the corresponding cell is marked as 1, otherwise it is left blank or empty. In general, all five groups do not have the same importance depending on the context of an analysis. However, in this study, we assume that the larger the number of members in a group is the more important the group is in the analysis. In the rest of the paper, we term the group importance as "*group weight*."

If we apply skyline query on our symbolic dataset, it will retrieve users "E" and "F" as key persons. This is because user "E" has the highest number of followees, "like" score, comment support, and the maximum group scores' sum among all the other persons. On the other hand, user "F" has the highest number of friends. Moreover, these two persons dominate the rest of the users.

Facebook data are increasing in an exponential manner, and nowadays it has become almost impossible to process such huge amounts of data in a single node computing system. Therefore, we apply *MapReduce* framework to speed up the computation and parallelism. *MapReduce* is a programming model and software framework, that was developed by *Google Inc*. Many real-world tasks are expressible in this model. Programmers find the system easy to use and hundreds of *MapReduce* programs have been implemented; and around one thousand *MapReduce* jobs are executed on Google clusters every day [3, 10, 23]. For better understanding and simplicity, we tried to keep the *MapReduce* explanation figures as simple as possible.

In summary, the contributions of this paper include the following aspects:

- ☐ We have considered effective utilization methods of skyline query to handle "Facebook data."
- ☐ We develop a novel scalable parallel algorithm to select the key person.
- ☐ We have empirically proved the efficiency of the proposed method through extensive experiments using synthetic datasets.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 presents the notions and properties of key person computation using skyline query. We explained the detailed algorithm with appropriate examples and analysis in Section 4. We experimentally evaluate the algorithm in Section 5 under a variety of settings. Section 6 concludes the paper.

2 Related Work

Retrieving key persons from social network includes the use of skyline computation. We review the related work on social network in Section 2.1. In addition, we survey existing methods for skyline computation in Section 2.2.

2.1 Social Networks

Discovering individuals who possess a given set of expertise, or who are familiar about a given topic, is a widely studied problem, usually known as the expert retrieval problem. The "expert finding task" was introduced by TREC in 2005. That task involved the exploration of an enterprise-data corpus (an email archive) and the retrieval of a set of individuals who are specialists in some given topic. DeMartini et al. have introduced a model for retrieving and ranking entities with its application to find experts [9]. The "expert team formation problem" is known as another characterization of the expert finding task. Such an approach uses the social associations among individuals, and the cumulative communication cost as the optimization term of the objective function [11]. Cao et al. have addressed the jury selection problem by using micro-blog services like Twitter to solve decision-making tasks [5]. In order to reduce the overall decision-making error rate, the authors have introduced two models for selecting jury members. They predict the error rate of each user by evaluating a Twitter graph.

Although several works discussed user profiling on social media [1] [15], to the best of our knowledge, our work is the first attempt that applies skyline query on Facebook, and it performs an extensive analysis of the performance on different online groups. Therefore, this paper complements the existing efforts for addressing key the person selection problem and creates an opportunity to recruit them as brand ambassador.

2.2 Skyline Query Processing

The use of skyline operator over large databases was first addressed by Borzsonyi *et al.* They introduced three methodologies: *Block Nested Loops* (BNL), *Divide and Conquer* (D&C), and *B-tree-based schemes* [4]. In BNL, each data object is compared with every other object in the database, and the object is reported only if no other object dominates it. For this purpose, a portion of main memory, called window W, is allocated. The input data are scanned sequentially and placed in W if it is not dominated by any object in W. A block of non-dominated objects is produced in every iteration. When the window is full, a temporary-file is used to store data that cannot be placed in W. This temporary-file is used as input to the next iteration. D&C divides the entire dataset into several blocks such that each block can fit into memory. Non-dominated objects for each individual block are then computed by a main-memory dominance check algorithm. The final result is obtained by merging the skyline objects for each block. Chomicki *et al.* improved BNL by presorting, that is, they introduced *Sort Filter Skyline* (SFS) as a variant of BNL [7]. While considering index-based approaches, Tan *et al.* addressed "Bitmap" and "Index" as two progressive skyline computing approaches [21]. In the "Bitmap" methodology, every dimensional value of a data point is exemplified by a few bits. By applying a bit-wise *AND* operation on these vectors, a given data point can be checked whether it is in the skyline. However, in "Index" methodology, a set of d-dimensional objects are organized into d lists. Lists are organized in such way that an object O is assigned to a list i if its value at dimension i is the best among all dimensions of O. Subsequently, lists are indexed by a B-tree, and the skyline is computed by perusing the B-tree. Perusing continues until an object that dominates the remaining entries in the B-trees is found. Currently, *Branch and Bound Skyline* (BBS) is known to be the most efficient method of computing skyline, and it was proposed by Papadias *et al*. BBS is a progressive algorithm and it is based on the *best-first nearest neighbor (BF-NN)* algorithm [18]. It directly prunes using the R*-tree structure. However, BF-NN deploys the technique of searching for nearest neighbor repeatedly.

Recently, more aspects of skyline computation have been explored. The k-dominant skyline was addressed by Chan *et al.* and they have proposed efficient ways to compute it in a high-dimensional space [6]. The n-of-N skyline query was proposed by Lin *et al.*, which was designed to perform online query on data streams, i.e., to find the skyline of the set comprising of the most recent n elements. If the datasets are too large and stored in a distributed environment, it may impossible to handle them in a centralized fashion [12]. Balke *et al.* first excavated skyline in a distributed environment by partitioning the whole dataset

User ID	Friend Power	Followee Strength	Like Score	Comment Support	Group Score
A	34	81	45	120	60
B	20	73	25	87	52
C	55	32	32	99	46
D	75	40	69	16	69
E	72	96	90	300	100
F	94	56	67	38	71
G	63	63	29	60	73
H	62	71	55	12	54
I	30	23	33	2	44
J	46	57	44	0	37

Table 1. Calculated Example Data

vertically [2]. Vlachou *et al.* introduced the concept of extended skyline set, which contains all data elements that are necessary to answer a skyline query in any arbitrary subspace [24]. Tao et al. discuss skyline queries in arbitrary subspace [22]. Other variants, like dynamic skyline [17] and reverse skyline [8] operators, have recently attracted substantial attention.

3 Preliminaries

In this section, we present definitions and basic properties of our key person selection problem. Let us assume that Table 1 shows the calculated values of the data described in Figure 1.

3.1 Social Network Metrics

At first, we introduce some definitions that are used in this work:

Definition 1 (Friend Power). Friend power, F_p, is the total number of friends that a user has on social media. It can be denoted as follows:

$$F_p = \mid \mathbf{Friends} \mid .$$

More friend counts indicate higher friend power. For example, in Table 1 user "E" has more friends (72) than user "B" (20); therefore, F_p of "E" is better than that of "B."

Definition 2 (Followee Strength). Followee strength, F_s, is the total number of followers that a user has on social media. It is denoted as follows:

$$F_s = \mid \mathbf{Followee} \mid .$$

More followers mean better followee strength. Table 1 illustrates that user "A" has better followee strength (81) than user "D" (40).

Definition 3 (Like Score). Like score, L_s, is the average number of "like" count that a user has achieved from his social media's posts or status updates. It is defined as:

$$L_s = \frac{\sum Like(ID_{post})}{\mid ID_{post} \mid}$$

where ID_{post} is a post or status update by UID and $Like(ID_{post})$ is the number of "likes" achieved by ID_{post}. From Figure 1, user "C" has posted a status update whose ID_{post} is "61" and it is "liked" by user "A." Let us assume that 37 other people has given a "like" to that post. Again, he/she has posted another status update whose ID_{post} is "65" and it is "liked" by user "E." Let us consider that 27 people have given a "like" to post "65." If we assume "C" has posted only two posts or updates, which are "61" and "65" and the total number of "likes" received by the two posts or updates are 64, then the "like" score of "C" is 32 $[\because (27 + 37) \div 2 = (64 \div 2) = 32]$.

Definition 4 (Comment Support). Friends, followers, and others may give comment feedback on some posts or status updates. The feedback can be neutral, positive, or negative. Comment support is the numerical summation of positive and negative feedbacks. If comment support is denoted by C_s, then

$$C_s = \sum CommentBias(Comment_n)$$

where $CommentBias(Comment_n)$ is a function that will return a numeric value of $Comment_n$. It returns $+1$ if the comment is positive, -1 if the comment is negative, and 0 if the comment is neutral or unrecognizable. In Figure 1, user "B" has posted a status update whose post ID is "27" and user "C" has replied with some text feedback: $Comment_2$. $Comment_2$ could be a positive, negative, or neutral sentence. Function $CommentBias(Comment_n)$ is responsible to find the bias of $Comment_2$. The return value could be $+1/-1/0$ depending on the bias of the comment. Let us assume that user "B" has received 175 positive, 88 negative, and 32 neutral comment feedbacks in his/her status updates. His/her comment support C_s becomes 87 [$\because C_s(B) = (+1) \times 175 + (-1) \times 88 + (0) \times 32 = 87$].

Definition 5 (Group Weight). A group with larger members has more importance than the ones with small members. It indicates the importance of a group on social network. However, for generalization, we use the normalized value of group member count for each group and call this measure "group weight" and denote it for group *"t"* as:

$$G_{wt}(t) = \frac{G_{counts(t)}}{\sum_{i=1}^{n} G_{counts(i)}}$$

where $G_{counts(i)}$ is the number of members in i^{th} group and n is total number of groups. $G_{counts(t)}$ denotes the number of members belonging to group *"t"* for which we are calculating the group weight. For example, if we assume that group "G1" has 300 members and the total members of different groups are $\sum_{i=1}^{n} G_{counts(i)} = 2413$ then after normalizing the G_{wt} value of "G1" becomes 12 [$\because G_{wt}(G1) = \{(300 \div 2413) \approx 0.12\} \times 100$]. Similarly, we assume that $G_{wt}(G2) = 25$, $G_{wt}(G3) = 15$, $G_{wt}(G4) = 17$ and $G_{wt}(G5) = 31$.

Definition 6 (Group Score). Group score is the summation of all group weights for a user of a social media that he/she belongs to.

$$G_s(U) = \sum_{i=1}^{n} G_{wt}(i) \times m(U, i)$$

where n is the total number of groups and $m(U, i)$ is a group membership factor. $m(U, i)$ is 1 if user U belongs to group i, otherwise $m = 0$. In Figure 1, user "A" is the member of groups $\{G1, G4$ & $G5\}$. Summing the group weights of those groups, we can get the group score of user "A," $G_s(A) = 12 + 17 + 31 = 60$, which is also shown in Table 1.

3.2 Dominance and Skyline

Let us assume we have a dataset DS (shown in Table 1) with five attributes. We represent friend power, followee strength, "like" score, comment support, and group score from (a_1) to (a_5), respectively. We use $U_i.a_k$ to denote the k^{th} $(1 \leq k \leq 5)$ attribute value of a user U_i, where i denotes user id (UID).

Definition 7 (Person Dominance). A user $U \in DS$ can dominate another user U' if user U's friend power, followee strength, "like" score, comment support, and group score are better or equal to user U''s and at least one of the mentioned feature of U is better than U''s. In Table 1 user "E" dominates user "B." This is because user "E" has more friends, followers, "like" score, comment support, and group score than user "B."

Definition 8 (Key Person Skyline). A user $U \in DS$ is in key person skyline of DS if U is not dominated by any other user in DS. In Table 1, users "E" and "F" are not dominated by any other user. Therefore, they represent the key person skyline result of the dataset DS.

3.3 Comment Bias

"Opinion mining," also known as "sentiment analysis," [14] is a research area for finding the bias of a comment. Opinions are important because they significantly influence our behaviors. Classification of opinion can be formulated as a supervised learning problem with three classes: positive, negative, and neutral. The features, which are often used in this problem, are listed below [16]:

Terms and their frequency. These features are individual words or word n-grams and their frequency counts. In some cases, we also consider word positions. These features could have been quite effective in sentiment classification.

Part of speech. Findings of numerous research work indicates that adjectives are significant indicators of opinions. Therefore, adjectives within a sentence have been treated as special features.

Opinion words and phrases. Opinion words are words that are commonly used to express positive or negative sentiments. For example, wonderful, beautiful, great, and amazing are positive opinion words, whereas poor, bad, and horrible are negative opinion words. Apart from individual words, there are also opinion idioms and phrases, e.g., a piece of cake. Opinion words and phrases are influential to sentiment analysis for obvious reasons.

In our proposed $CommentBias()$ function, *opinion words and phrases* has been considered as key bias detection approach.

Rules of opinions. Although opinion words and phrases are important, there are also many other expressions that contain no opinion words or phrases but they indicate opinions or sentiments.

Negations. Negation words are crucial because their presence often change the orientation of the opinion. For example, the sentence "I don't like this book" is negative. However, negation words must be handled with extra care because occurrences of such words do not confirm a negative meaning. For example, the "not" in "not only but also" does not change the orientation direction.

Syntactic dependency. Words dependency-based features generated from parsing or dependency trees are also used by several researchers.

3.4 Hadoop MapReduce

Hadoop is known to be an open source implementation of Google Inc.'s *MapReduce* framework. It is maintained by Apache Software Foundation. This framework is designed to allow users to submit a *MapReduce* job by defining the *map* and *reduce* functions. Data are represented as $\langle key, value \rangle$ pairs and computation is distributed across the cluster of autonomous machines. Two user-defined functions, called *Map* and *Reduce*, are responsible in performing computational tasks:

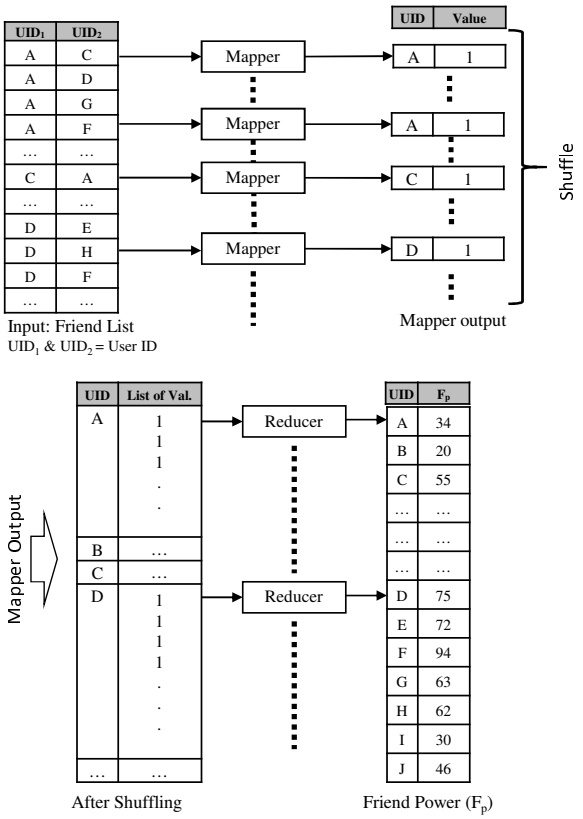
$$Map(k_1, v_1) \rightarrow list(k_2, v_2)$$
$$Reduce(k_2, list(v_2)) \rightarrow list(v_3)$$

The *Map* function (sometimes called *Mapper*) processes on each $\langle key, value \rangle$ pair of input data, and produces intermediate $\langle key, value \rangle$ pairs. The intermediate $\langle key, value \rangle$ pairs are then sorted and grouped associated with the same key. The *Reduce* function (sometimes called *Reducer*) takes a key and a list of values for that key, applies the processing logic, and generates the final result as a list of values.

4 Key Person Finding Algorithm

Our *MapReduce* based key person finding algorithm has the following seven consecutive calculation phases:

1. Friend power (F_p)
2. Followee strength (F_s)
3. "Like" score (L_s)
4. Comment support (C_s)
5. Group weight (G_{wt})



Algorithm: Friend Power Calculation

1. ***Function MAPPER (key, value)***
   key: $UID_1 - user\ id\ (string)$,
   $value$: $UID_2 - friend's\ id\ (string)$
2. ***Begin***
3. $Process\ input\ string$: $< UID_1, UID_2 >$ *and depict* $< UID_1, 1 >$ *as Mapper output*
4. ***End***
5. ***Function REDUCER (key*** $, < list\ of\ values >)$
   key: $UID - user\ id\ (string)$,
   $list\ of\ values$: $list\ of\ 1's\ (integer\ items)$
6. ***Begin***
7. $set\ F_p = 0$
8. *for each item in* $< list\ of\ values >$ *do*
9. $val = fetch\ item\ from < list\ of\ values >$
10. $F_p = F_p + val$
11. $end\ loop$
12. write UID, F_p to $HDFS$
13. ***End***

Figure 2. Friend power computation procedure

6. Group score (G_s)
7. Sorting and skyline computation

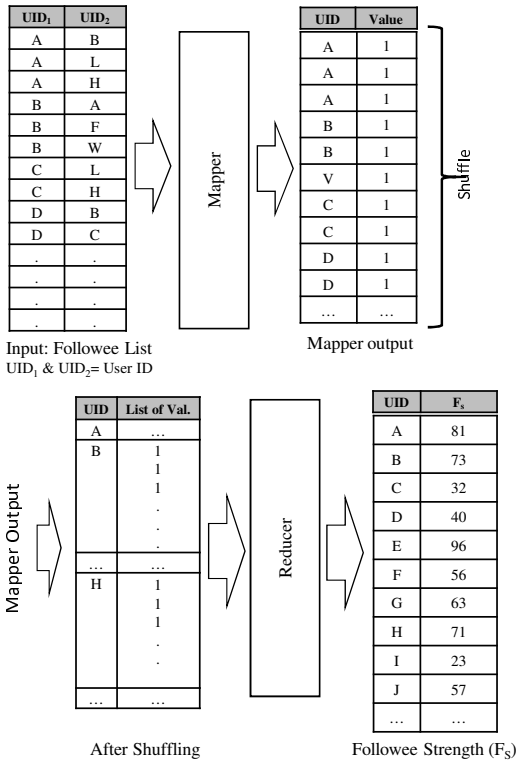
4.1 Friend Power (F_p)

Calculating the friend power (F_p) in *MapReduce* fashion is the first phase of algorithm. We assume that our friend list *DataSet* is in *kvs* ($key\ value\ storage$) format and structured as: $\langle UID_1, UID_2 \rangle$, where UID_2 is a friend of UID_1. In addition these data are distributed among several *DataNodes*. When a *Mapper* reads $\langle UID_1, UID_2 \rangle$ pair, it depicts one $\langle key, value \rangle$ pair as the intermediate result (*i.e., Mapper output*): $\langle UID_1, 1 \rangle$ indicating that UID_1 has one friend. According to *MapReduce* framework, the *Mapper* output is shuffled; group by corresponding *keys*, and *values* are tagged together as $list(values)$. In the proposed case, the $list(values)$ is represented by a sequence of $1's$. After shuffling and grouping, data with the same *key* are fed into a single *Reducer*. The *Reducer* counts the number of $1's$ in the $list(value)$ sequence and produces the counting result as our F_p.

Figure 2 represents the F_p computation procedure with its formal algorithm. For the first input pair $\langle A, C \rangle$ the *Map* worker produces one $\langle key, value \rangle$ pair: $\langle A, 1 \rangle$. This is because user "A" has one friend "C," and therefore, the friend power of "A" increases by one. By applying (UID, $count(value)$) each *Reduce* worker produces the total friends number (which we call friend power F_p). For example, user "A" has 34 friends, user "D" has 75 friends, etc.

4.2 Followee Strength (F_s)

This section explains followee strength (F_s) calculation in *MapReduce* framework. This calculation is similar with F_p calculation. In this case, we assume that our follower list *DataSet* is in kvs format and structured as: $\langle UID_1, UID_2 \rangle$, where UID_2 is following UID_1. As stated earlier, this *DataSet* is distributed among several



Algorithm: Followee Strength Calculation

1. ***Function MAPPER (key, value)***
   key: $UID_1 - user\ id\ (string)$,
   $value$: $UID_2 - followee's\ id\ (string)$
2. ***Begin***
3. $Process\ input\ string$: $< UID_1, UID_2 >$ *and depict* $< UID_1, 1 >$ *as Mapper output*
4. ***End***
5. ***Function REDUCER (key*** , $<$ ***list of values*** $>$***)***
   key: $UID - user\ id\ (string)$,
   $list\ of\ values$: $list\ of\ 1's\ (integer\ items)$
6. ***Begin***
7. $set\ F_s = 0$
8. *for each item in* $< list\ of\ values >$ *do*
9. $val = fetch\ item\ from < list\ of\ values >$
10. $F_s = F_s + val$
11. $end\ loop$
12. write UID, F_s to $HDFS$
13. ***End***

Figure 3. Followee strength computation procedure

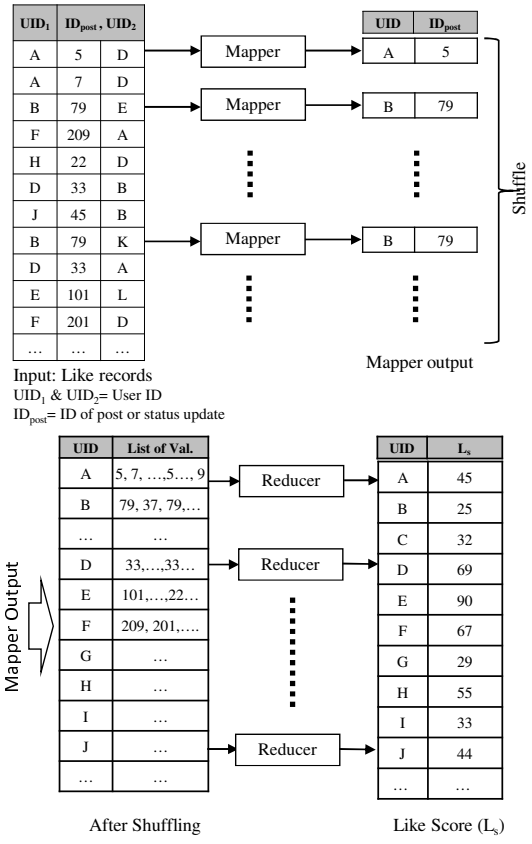
$DataNodes$. When a *Mapper* reads $\langle UID_1, UID_2 \rangle$ it knows that UID_2 is following UID_1. Therefore, *Mapper* output depicts a pair of value $\langle UID_1, 1 \rangle$, indicating that UID_1 has one follower. According to *MapReduce* framework, the *Mapper* output is shuffled, grouped by *keys*, and the corresponding *values* are tagged together as $list(values)$. Like the F_p calculation, the $list(values)$ is represented by the sequence of $1's$. After shuffling and grouping, data with the same *key* are fed into a single *Reducer*. The *Reducer* counts the number of $1's$ in the $list(value)$ and produce the counting result as our F_s.

Figure 3 illustrate the followee strength computation process. Where for the first input pair $\langle A, B \rangle$ *Mapper* produces $\langle key, value \rangle$ pair $\langle A, 1 \rangle$ which means "A is followed by another user B." Here "B is followed by A" is not true. Subsequently, by applying (UID, $count(value)$) each *Reducer* produces the total followee number (which we termed as followee strength F_s); for example, user "B" has 73 followers, user "H" has 71 followers, etc.

4.3 Like Score (L_s)

"Like" score (L_s) is one of the important matrices of a social network like Facebook. This section explains the procedure of calculating L_s in *MapReduce* framework. Here, we assume that the kvs format of our input data is structured as: $\langle UID_1, (ID_{post}, UID_2) \rangle$, where UID_1 represents a user id that posted the original post or status update, ID_{post} is the id of the post posted by UID_1, and UID_2 is the id of the person who gave "like" to the ID_{post}. After reading $\langle UID_1, (ID_{post}, UID_2) \rangle$ pair, a *Mapper* confirms that user UID_1 receives a "like" for his/her post ID_{post}. Therefore, it depicts a $(key, value)$ pair as: $\langle UID_1, ID_{post} \rangle$. As we discussed previously, these *values* are shuffled, grouped together, and *values* with similar *key* will be fed to a single *Reducer*. The $list(values)$ would be a collection of ID_{post} in which UID_1 has achieved "likes" from other users. After shuffling, a single *Reducer* calculates the average "likes" that user UID_1 has achieved. The calculated result is known as "like" score (L_s).

Figure 4 illustrates the "like" score (L_s) computation process with its formal algorithm. The figure shows that when a *Mapper* reads $\langle A, 5, D \rangle$ as input, it depicts $\langle A, 5 \rangle$ as the intermediate output (*i.e. Mapper output*). Similarly, when a *Mapper* reads $\langle B, 79, E \rangle$ it produces $\langle B, 79 \rangle$. After shuffling, each *Reducer* receives values



Algorithm: Like Score Calculation

1. ***Function MAPPER (key, value)***
   $key: UID_1 -$ *user id (string)*, $value: [ID_{post}, UID_2]$
   ID_{post} – *social media post id(string)*
   UID_2 – *user id (who liked the post)*
2. ***Begin***
3. *Process input string:* $< UID_1, [ID_{post}, UID_2] >$ *and depict* $< UID_1, ID_{post} >$ *as Mapper output*
4. ***End***
5. ***Function REDUCER (key,*** $<$ ***list of values*** $>$***)***
   $key: UID$ – *user id (string)*
   $list\ of\ values$: *list of* ID_{post}
6. ***Begin***
7. $post\_id\_list = new\ ArrayList < string >$
   $like\_count = 0,\ post\_id\_count = 0$
8. *for each item in* $<$ *list of values* $>$ *do*
9. *post_id = fetch item from* $<$ *list of values* $>$
10. *if (!post_id_list.contains(post_id)) then*
11. $post\_id\_count++$
12. $post\_id\_list.add(post\_id)$
13. *end if*
14. $like\_count++$
15. *end loop*
16. $L_s = like\_count \div post\_id\_count$
17. $write\ (UID, L_s) to\ HDFS$
18. ***End***

Figure 4. Like Score (L_s) calculation procedure

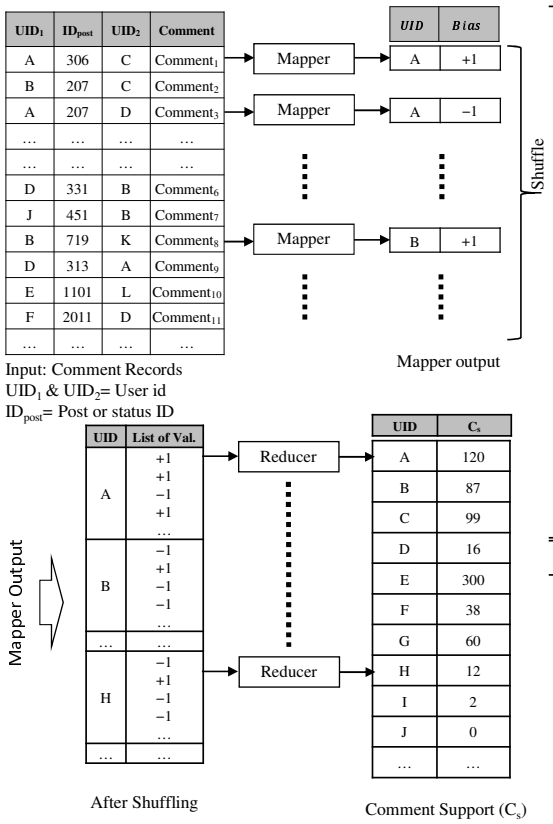
with similar UID. In Figure 4, the first *Reducer* gets the $list(values) = 5, 7, \cdots 5, \cdots, 9 \cdots$ and it calculates the average of like score L_s. For example, the L_s value for user "A" is 45, for user "B" it is 25, etc.

4.4 Comment Support (C_s)

Comment support (C_s) is another key matrix of social media. Users post status update or comments in social media. Other people reply or send feedback on those status update or comments. Comment feedback may be neutral, positive or negative. Some feedback bias is too hard to understand due to their complexity. For simplicity, we consider those complex feedbacks as neutral. Let us assume that we have a function named $CommentBias()$ that determines whether the parameter comment feedback is positively or negatively biased. Let us also assume input data kvs format to be structured as: $\langle UID_1, (ID_{post}, UID_2, Comment_{fdbk})\rangle$. Where UID_1 represent original status posted by the person, ID_{post} is the post id, and UID_2 is the user id that provides the feedback. $Comment_{fdbk}$ is the plain text comment feedback. When *Mapper* reads $\langle UID_1, (ID_{post}, UID_2, Comment_{fdbk})\rangle$, it uses the $CommentBias()$ function to get the bias of comment feedback. $CommentBias(Comment_{fdbk})$ returns $+1$ or -1 depending on $Comment_{fdbk}$ being either a positive or negative comment. This function also returns 0 for neutral or complex comment, whose bias is hard to understand. After processing each input data, each *Mapper* retrieves $\langle UID_1, \pm 1\rangle$, indicating UID_1 receives positive or negative feedback. The *Mapper* does not produce any result if the $CommentBias()$ returns 0 and has no significance in the calculation of C_s.

After shuffling and grouping based on *key* similarity, the $list(values)$ will be a sequence of $+1$ and -1. Subsequently, each *Reducer* calculates the numerical aggregation (*e.g., sum*) of those values and produces the result as comment support (C_s).

Figure 5 illustrates the process of calculating C_s. When a *Mapper* reads $\langle A, 306, C, Comment_1\rangle$, then it tries to find the bias of $Comment_1$ using $CommentBias()$ function. Let us assume $Comment_1$ is a



Algorithm: Comment Support Calculation

1. ***Function MAPPER(key*: UID_1, *value*: [*IDpost*, UID_2, *Comment*])**
   UID_1 : *user id who give the post(string)*
   UID_2 : *user id who reacted to the post(string)*
   Id_{post}: *post id(string)* & *Comment: text comment(string)*
2. ***Begin***
3. *depict* $< UID_1, CommetntBias(Comment)) >$ *as Mapper output*
4. ***End***
5. ***Function REDUCER (key,* $<$ *list of values* $>$*)***
   key: UID $-$ *user id* & *list of values: list of* ± 1 *or* 0
6. ***Begin***
7. $C_s = 0$
8. *for each item in* $<$ *list of values* $>$ *do*
9. $\quad bias\_val =$ *fetch item from* $<$ *list of values* $>$
10. $\quad C_s = C_s +$ *bias_val*
11. *end loop*
12. *write* (UID, C_s) *to HDFS*
13. ***End***

Algorithm: CommentBias

1. ***Function CommentBias (Comment as string)***
2. ***Begin***
3. *determine Comment bias using opinion mining techniques – Opinion word & phrase*
4. *if* $(Comment\ is\ positive\ sentence)$ *return* $+1$
5. *elseif* $(Comment\ is\ negative\ sentence)$ *return* -1
6. *else return* 0
7. *end if*
8. ***End***

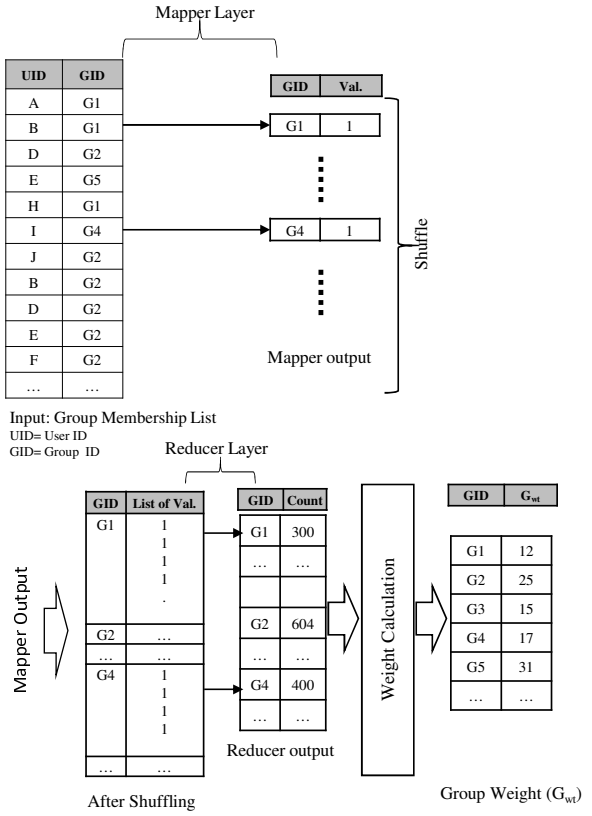
Figure 5. Comment support (C_s) calculation procedure

positive comment, and therefore, $CommentBias()$ will return $+1$. In this particular scenario, the *Mapper* output becomes $\langle A, +1\rangle$. Similarly for input $\langle A, 207, C, Comment_3\rangle$, mapper outputs $\langle A, -1\rangle$ (assuming $Comment_3$ is a negative comment). After shuffling, grouping, and feeding into *Reducer* the Comment Support (C_s) is being calculated as 120 for user "A," 87 for user "B," etc. For better understanding, we have included formal algorithm with in Figure 5.

4.5 Group Weight (G_{wt})

To calculate group weight, each mapper takes UID and their corresponding group lists as input. Each Mapper generates a pair of intermediate output: $\langle GID, value\rangle$. Here, GID represents group id. After shuffling and grouping by the corresponding GID, the *Mapper*s outputs are sent to *Reducer*s. Each *Reducer* produces $\langle GID, count(value)\rangle$ pair for each group, where $count(value)$ is the membership number of each group. Finally, a separate module named *WeightCalculation* calculates the normalized value of membership count. These normalized values are known as Group Weight (G_{wt}).

Figure 6 shows the procedure of group weight computation. For the input pair $\langle A, G1\rangle$, *Mapper* produces $\langle G1, 1\rangle$. Subsequently, using $\langle GID, count(value)\rangle$ pair each *Reducer* produces the total membership number for each group. For example, group "G1" has 300 members, "G2" has 604 members, etc. After normalizing the G_{wt} value of "G1," it becomes 12 $[\because G_{wt}(G1) = (300 \div 2413) \rightarrow 0.12 \times 100, 2413 = \sum_{i=1}^{n} G_{counts(i)}]$ and the G_{wt} value of "G2" becomes 25, etc.



Algorithm: Group Weight Calculation (MapReduce part)

1. ***Function MAPPER (key, value)***
   key: $UID - user\ id\ (string)$,
   $value$: $GID - group\ id\ (string)$
2. ***Begin***
3. $Process\ input\ string: < UID, GID >$ *and depict* $< GID, 1 >$ *as Mapper output*
4. ***End***
5. ***Function REDUCER (key*** $, < list\ of\ values >$***)***
   $key: GID - group\ id\ (string)$,
   $list\ of\ values$: $list\ of\ 1's\ (integer\ items)$
6. ***Begin***
7. $set\ tG_{wt} = 0$
8. *for each item in* $< list\ of\ values >$ *do*
9. $val = fetch\ item\ from < list\ of\ values >$
10. $tG_{wt} = tG_{wt} + val$
11. $end\ loop$
12. *write* GID, tG_{wt} to $HDFS$
    (Weight Calculation module will use these tG_{wt} *values to get normalized* G_{wt}*)*
13. ***End***

Figure 6. Group weight computation procedure

4.6 Group Score (G_s)

In this *MapReduce* procedure each *Mapper* takes $\langle UID, GID\rangle$ pair and G_{wt} as input and generates pairs: $\langle UID, value\rangle$. Where $value$ is the normalize weight value for each group. On the downside, after shuffling each *Reducer* produces the $\langle UID,\ sum(value)\rangle$ pair for each user. We termed $sum(value)$ as group score (G_s).

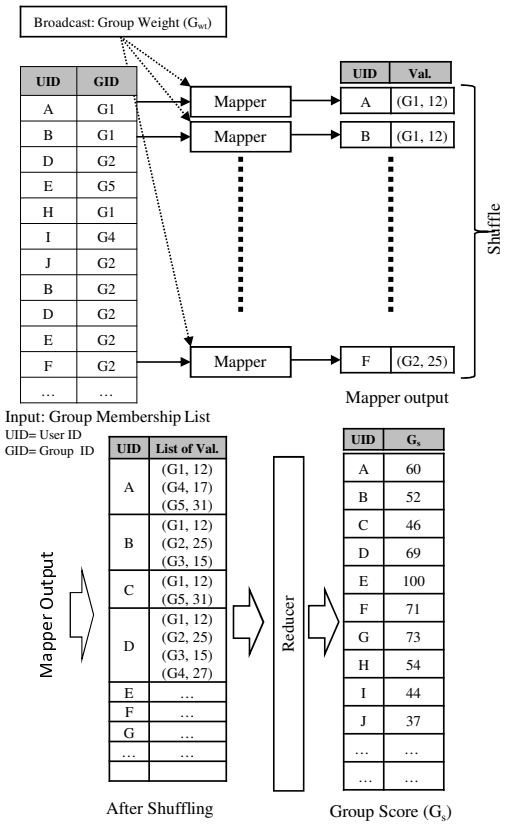
Group score computation process is shown in Figure 7. For the input pair $\langle A, G1\rangle$, *Mapper* produces pair $\langle A, (G1, 12)\rangle$. Here normalized group weight *value* for group "G1" is 12. Subsequently, using $sum(value)$ function, each *Reducer* produces the group score for each user. To illustrate, user "A" has a group score of 60, user "B" has a group score of 52, etc. For better understanding, we have included a formal algorithm in Figure 7

4.7 Sorting and Skyline Computation

We perform descending sort on UID according to F_p, F_s, L_s, C_s, and G_s. A similar procedure has been applied for these five sorting. To avoid redundancy in this paper, we discuss only about the sorting procedure based on F_p. Initially, each *Mapper* takes $\langle UID, F_p\rangle$ pair as input and produces $\langle F_p, UID\rangle$ pair. After completing the shuffling process on F_p, all $\langle F_P, UID\rangle$ pairs are sent as *Reducer*s' input. Subsequently, each *Reducer* outputs UID in descending order based on F_p.

Figure 8 represents this sorting procedure. *Mapper* reverses the input pair $\langle A, 34\rangle$ as $\langle 34, A\rangle$. After sorting on friend power in descending order each *Reducer* outputs sorted UID. In Figure 8, user "F" holds the topmost position because of his/her highest F_p. It is to be noted that to sort in descending order, we must override the default output key class of *Hadoop* (because the default sorting operation of *Hadoop* framework is ascending). For clear understanding, we have included a formal algorithm in Figure 8.

In the next stage, the proposed method receives five sorted UID lists respectively on F_p, F_s, L_s, C_s, and G_s. We must select our key person based on these five criteria. That means a key person has at most five



Algorithm: Group Score Calculation

1. Broadcast G_{wt} to each *Mapper*
2. ***Function MAPPER*** $(\boldsymbol{key}, \boldsymbol{value})$
   key: UID − *user id (string)*
   $value$: GID − *group id (string)*
3. ***Begin***
4. $process < UID, GID >$ *and*
   depict $< UID, (GID, G_{wt}) >$ *as Mapper output*
5. ***End***
6. ***Function*** **REDUCER** $(\boldsymbol{key}, < \boldsymbol{list\ of\ values} >)$
   key: UID − *user id (string)*
   $list\ of\ values$: $list\ of\ [GID, Gwt]$
   GID − *group id (string)* & Gwt − *group weight (number)*
7. ***Begin***
8. $set\ G_s = 0$
9. *for each item in* $< list\ of\ values >$ *do*
10. $val = fetch\ item.\ G_{wt}\ from < list\ of\ values >$
11. $G_s = G_s + val$
12. *end loop*
13. $write\ UID, G_s\ to\ HDFS$
14. ***End***

Figure 7. Group score computation procedure

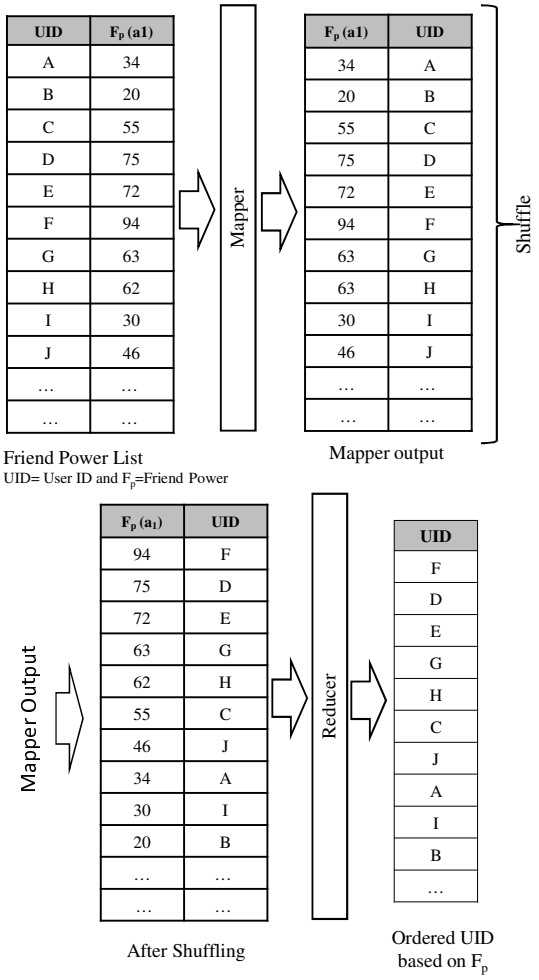
times the opportunity to be selected as the best person. Therefore, our method maintains a counter for each user and if a user is retrieved five times, then it stops the candidate selection.

Figure 9 shows the skyline computation procedure. In the first iteration, it selects users "F" and "E" as candidates and sets the frequency counter value 1 for "F" and 4 for "E" [as it picks $\{F, E, E, E, E\}$ in the first iteration]. Subsequently, it chooses user "D," "A," and "G" as candidates and sets their corresponding frequency counter values. Thereafter, the counter value for user "E" becomes 5, which is equivalent to the total number of input criterion. Finally, the candidate list for key persons are $\{A, D, G, E, F\}$. Now, we can easily compute skyline by comparing these candidates. The details of this skyline computation procedure are discussed in [20]. For interested audiences, we briefly describe the idea of that work below.

In the work [20], we partitioned the dataset vertically and sorted each partition. That means, we have to sort the data objects according to domain values. Based on the result of sorting, the object *ID*s are given a ranking value. An *Eliminator module* collects top *ID*s from each domain horizontally. The *module* maintains a counter for each retrieved object. When a counter value becomes equal to the number of domains, then it stops *ID* collection. The IDs collected by the *module* are candidates of *Skyline* query result.

In Figure 9, *UIDs* are already sorted according to *friend power, followee strength, like score, comment support* and *group score*. In the first iteration, the *Eliminator module* picks $\{E, F, E, E, E\}$, as they are the top ranked *UIDs*. At the same time, the *Eliminator module* maintains a counter for each retrieved *UID*. After the first iteration, the counter of *UID* "E" is set to 4 as it occurs four times in the retrieved list. For the same reason, the counter for *UID* F is set to 1. In the second iteration, $\{D, A, D, A, G\}$ are picked and the counters are set or updated (if needed). In the third iteration, the *UID* picking procedure stops as the *Eliminator module* picks "E," updates the counter value for "E," and finds that it is equal to 5, which is the same as the number of domains.

When the *Eliminator module* stops, it already has had a list of *UIDs*. In the example, the list contains $\{F, E, D, A, G\}$ − known as *candidate list*. Now, each of the elements in the *candidate list* has its own domain values.



Algorithm : Ordering with MapReduce

1. ***Function MAPPER*** $(\boldsymbol{key}, \boldsymbol{value})$
   $key: UID - user\ id\ (string)$
   $value: F_p$ *–friend power (number)*
2. ***Begin***
3. $process < UID, F_p > and$
   depict $< F_p, UID >$ *as Mapper output*
4. ***End***
5. ***Function*** **REDUCER** $(\boldsymbol{key}, < \boldsymbol{list\ of\ values} >)$
   $key: F_p - friend\ power\ (number)$
   $list\ of\ values: list\ of\ UID$
6. ***Begin***
7. *for each item in* $< list\ of\ values >$ *do*
8. $fetch\ UID\ from < list\ of\ values >$
9. $write < UID > to\ HDFS$
10. *end loop*
11. ***End***

Figure 8. Sorting on friend power

In the example, the domain value of "F" for five domain F_p,F_s, L_s, C_s & G_s are $\{94, 56, 67, 38$ & $71\}$, respectively. Table 2 shows us the domain values for all UID in the candidate list:

uid	F_p	F_s	L_s	C_s	G_s
F	94	56	67	38	71
E	72	96	90	300	100
D	75	40	69	16	69
A	34	81	45	120	60
G	63	63	29	60	73

Table 2. Candidates' Domain Values

It is obvious, when we perform *dominance tests* among those *candidate list UIDs*, "E" will dominate "A" and "G." No one within the *candidate list* dominates "F," "E," and "D," therefore they are our desired key persons. The *dominance test* operations are performed by the *Simple Comparison module*.

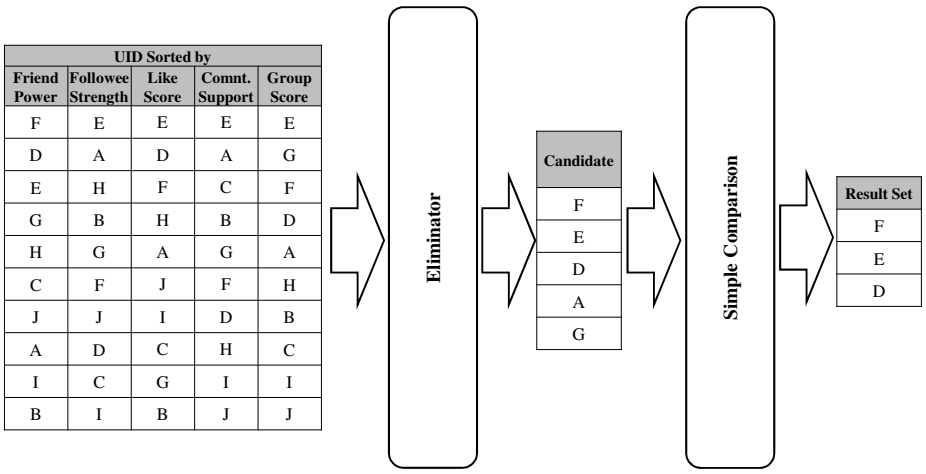


Figure 9. Skyline computation

5 Performance Evaluation

This section reports our experimental results to validate the effectiveness and efficiency of the proposed method. We set up a cluster of four commodity PCs in a high-speed gigabit networks, each of which had an Intel Core 2 Duo E8500 3.16 GHz CPU, with 8 GB memory. These machines were connected with *Cisco SG300-20* gigabit manageable switch. We compile the source codes under Java V8. We used *hadoop* version 2.5.2 and the OS platform was 64 bit CentOS 7. The replication parameter of *hadoop* configuration was 2.

One of the important goal for designing our experiments was to study the flexibility of processing large amount of data using the proposed algorithm in *MapReduce* framework. To study the effectiveness, we have compared our proposed method with *"Single Node"* execution. The term "*Single Node*" is used to specify a standalone autonomous desktop PC. It is neither a part of the *MapReduce* framework nor it is considered as a part of any other cluster or grid. It is used to study the performance variation of our proposed algorithm while not using *MapReduce* framework. We have also conducted experiments with the domain value idea expressed in [19], where a similar problem is considered as the graph mining problem. To conduct experiments, we used synthetic datasets (because of the unavailability of Facebook data); each experiment is repeated five times and the average result is considered for performance evaluation.

5.1 Effect of Proposed Algorithm in *MapReduce*

We study the effect of various steps described in Section 4. Figure 10 (a−f) shows the effect of F_p, F_s, L_s, C_s, G_{wt} and G_s calculation. In general, social media mining problems are considered as graph-mining problem. Similar graph mining problems are basically analogous to *top-k* query problem [*e.g., ranking problem*], rather than *skyline* query. However, a *top-k* query requires users to have the domain knowledge, while for *skyline* query, no domain knowledge is required. We have compared the performance of our proposed algorithms with the *skyline* idea described in [19], where the problem of InfraSky is explained as a graph-mining issue and domain values are expressed by *indegree* and *outdegree* of a node. For example, in case of F_p calculation, we assumed that if user "A" has a friend "B" then there exists a directed edge from node "B" to "A" and so on for other metrics. From each of the experimental results, shown in Figure 10, we can see that the performance of using *MapReduce* is better than the performance of Single Node implementation as well as the InfraSky idea defined in [19]. However, we can also observe in every experiment that the execution time of using *MapReduce* framework is almost identical (almost a constant value), even if the cardinality of data set increased significantly. This identical execution time indicates that the proposed $MapReduce$-based method can be used to efficiently process larger amounts of data than in our experiments. Meanwhile, the other methods are not suitable for processing large scale of data, as their execution time increases linearly.

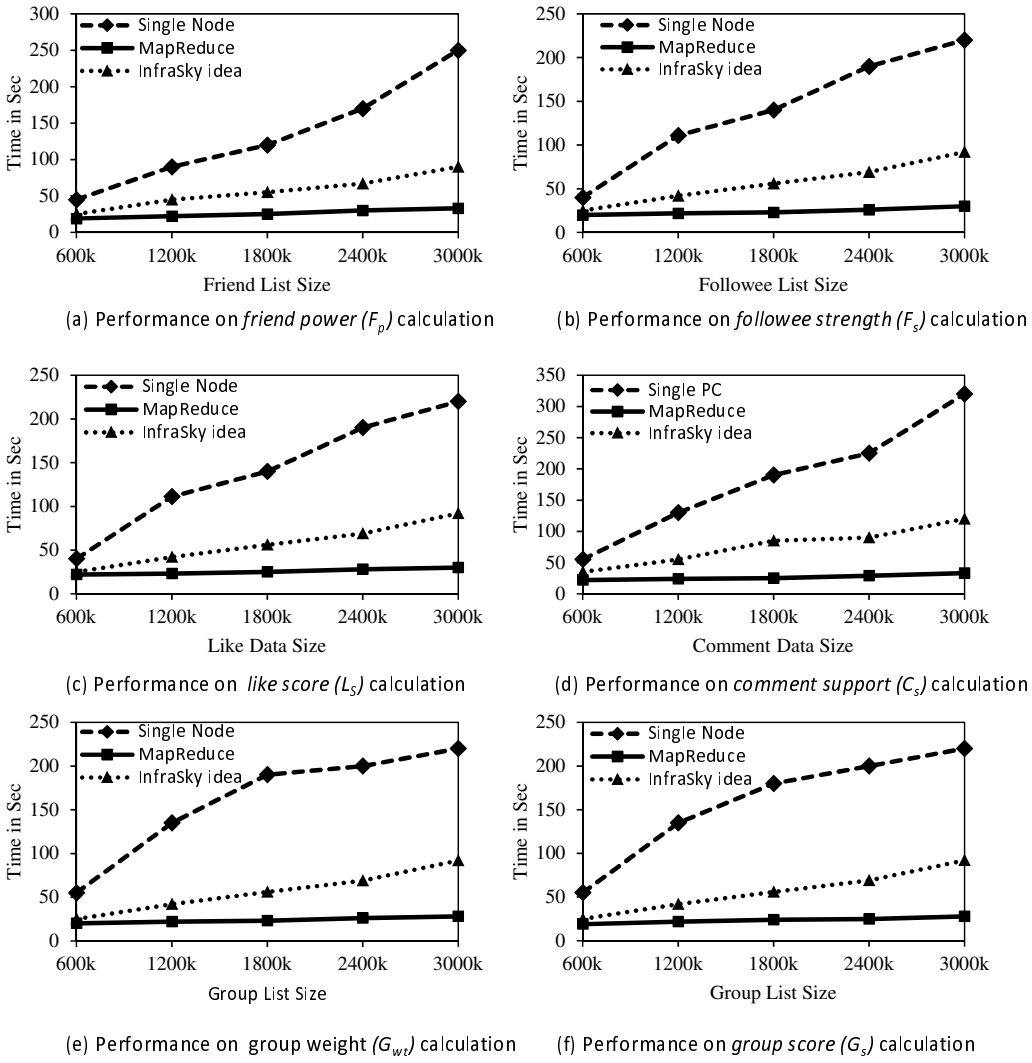


Figure 10. Performance of domain value calculation

5.2 Effect of Skyline Computation

The naive method of skyline computation is a greedy approach and it requires a lot of computational resources like memory, and CPU time, etc. We report the performance on skyline computation. For this experiment, the data cardinality varies from 50k to 800k. The performance result is illustrated in Figure 11. In traditional way the complexity of "single node" skyline computation is $O(n^2 - 1)$. It is observed that "single node"-based method is highly affected by cardinality. If the data size increases more than 100k ,it cannot compute the final result due to memory space limitations, and this is because of the large cardinality of non-dominating records. However, our proposed algorithm does not face such a problem. The implementation of dominance test portion of the idea [19] was implemented in our proposed *MapReduce* framework, therefore, the execution time is identical.

5.3 Effect of New Metrics

The major problem of using *skyline* query is that it may produce a "*too few or too large*" result set. When the result set is too small, the user may not get any advantage from the computation as the resultant data set may have been already occupied or may not be interested to serve. When the result set is too large, it may also confuse the result seeker in making any choice. Expanding the size of social media matrices

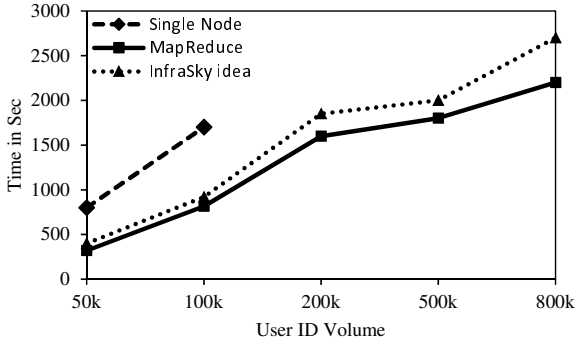


Figure 11. Performance on skyline computation

work can minimize the possibility of encountering such a problem. In our previous work [25], we have used three social media matrices: friend power (F_p), followee strength (F_s) and group score (G_s). However, in this paper work we have introduced two new metrics: "like" score (L_s) and comment support (C_s). The enhancement of result due to upgrade of social media matrices has been shown in Figure 12. It is clear that if we use the conventional three dimensional approach [25], we may have very few results needed to select key persons. Meanwhile, the proposed five dimensional approach gives us a better opportunity to choose the perfect ones.

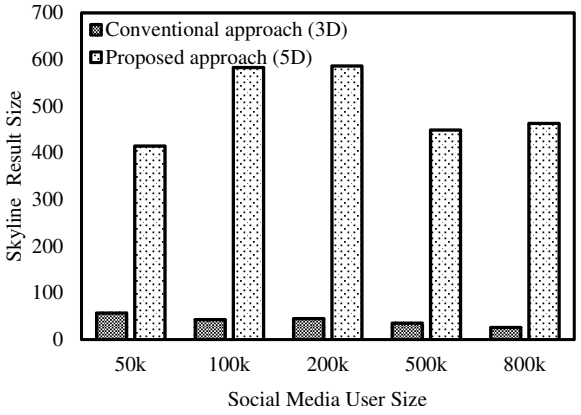


Figure 12. Effect of introducing two new Social Media metrics

6 Conclusion

In this study, we addressed the problem of selecting key persons from different groups of Facebook network. We consider a novel algorithm to identify key persons. The main feature of the proposed algorithm is that it can retrieve results using the skyline query. Moreover, in our proposed approach we consider the parallel distributed *MapReduce* framework to speed up the computation process and to handle massive data. Extensive experiments demonstrate the efficiency of our algorithm for synthetic datasets.

It is noteworthy to mention that this work can be expanded in a number of directions. First, to generate more precise results we need to consider the regular activities of people in social networks such as share, check-in etc. Secondly, if the result is too high or too low, management may be confused to select the key person. In such case we need to consider other variant queries such as representative skyline query and top-k query.

Acknowledgment

This work is supported by KAKENHI (16K00155, 23500180, 25.03040) Japan. Asif Zaman is under the Japanese Government MEXT Scholarship program. Annisa is under the Indonesian Directorate General of Higher Education (DIKTI) Scholarship program.

References

[1] A. Alkouz, E. W. D. Luca, and S. Albayrak. Latent semantic social graph model for expert discovery in facebook. In *Proceedings of 11th Int'l Conference on Innovative Internet Community Systems (IICS)*, pages 128–138, 2011.

[2] W-T. Balke, U. Gntzer, and J-X. Zheng. Efficient distributed skylining for web information systems. In *Proceedings of EDBT*, pages 256–273, 2004.

[3] S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y. Tian. A comparison of join algorithms for log processing in mapreduce. In *Proceedings of SIGMOD*, pages 975–986, 2010.

[4] S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proceedings of ICDE*, pages 421–430, 2001.

[5] C. C. Cao, J. She, Y. Tong, and L. Chen. Whom to ask? jury selection for decision making tasks on micro-blog services. *Proceedings of the VLDB Endowment*, 11(2):1495–1506, 2012.

[6] C. Y. Chan, H. V. Jagadish, K-L. Tan, A. K. H. Tung, and Z. Zhang. Finding k-dominant skyline in high dimensional space. In *Proceedings of ACM SIGMOD*, pages 503–514, 2006.

[7] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *Proceedings of ICDE*, pages 717–719, 2003.

[8] E. Dellis and B. Seeger. Efficient computation of reverse skyline queries. In *Proceedings of VLDB*, pages 291–302, 2007.

[9] D. Gianluca, G. Julien, and N. Wolfgang. A vector space model for ranking entities and its application to expert search. *Advances in Information Retrieval*, 5478:189–201, 2009.

[10] D. Jiang, A. K. H. Tung, and G. Chen. Map-join-reduce: Toward scalable and efficient data analysis on large clusters. *IEEE Transactions Knowledge Data Engineering (TKDE)*, pages 1299–1311, 2011.

[11] T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *Proceedings of the ACM SIGKDD*, pages 467–476, 2009.

[12] X. Lin, Y. Yuan, W. Wang, and H. Lu. Stabbing the sky: Efficient skyline computation over sliding windows. In *Proceedings of ICDE*, pages 502–513, 2005.

[13] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: the k most representative skyline operator. In *Proceedings of ICDE*, pages 86–95, 2007.

[14] Bing Liu. Ch-11: Openion mining and sentiment analysis. In *book Web Data Mining -Exploring Hyperlinks, Contents,and Usage Data*. Springer, New York, 2011.

[15] H. Liu. Social network profiles as taste performances. *J. Computer-Mediated Communication*, 13(1):252–275, 2009.

[16] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Found. Trends Inf. Retr.*, 2(1-2):1–135, January 2008.

[17] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *Proceedings of SIGMOD*, pages 467–478, 2003.

[18] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Transactions on Database Systems*, pages 41–82, 2005.

[19] Zhuo Peng, Chaokun Wang, Lu Han, Jingchao Hao, and Xiaoping Ou. *Discovering the Most Potential Stars in Social Networks with Infra-skyline Queries*, pages 134–145. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[20] Md. A. Siddique, H. Tian, and Y. Morimoto. Distributed skyline computation of vertically splitted databases by using mapreduce. In *Proceedings of DASFAA Workshop*, pages 33–45, 2014.

[21] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *Proceedings of VLDB*, pages 301–310, 2001.

[22] Y. Tao, X. Xiao, and J. Pei. Subsky: Efficient computation of skylines in subspaces. In *Proceedings of ICDE*, pages 65–65, 2006.

[23] R. Vernica, M. J. Carey, and C. Li. Efficient parallel set-similarity joins using mapreduce. In *Proceedings of SIGMOD*, pages 495–506, 2010.

[24] A. Vlachou, C. Doulkeridis, Y. Kotidis, and M. Vazirgiannis. Skypeer: Efficient subspace skyline computation over distributed data. In *Proceedings of ICDE*, pages 416–425, 2007.

[25] Asif Zaman, Md. Anisuzzaman Siddique, Annisa, and Yasuhiko Morimoto. Selecting key person of social network using skyline query in mapreduce framework. In *Proceedings of the International Symposium on Computing and Networking (CANDAR)*, pages 213–219, 2015.