Study on Secure MapReduce Computation of Representative Object Selection from Big Data

（ビッグデータからの代表的オブジェクト選別の安全な MapReduce計算法に関する研究）

by

Asif Zaman

A dissertation submitted

Graduate School of Engineering, Hiroshima University

in partial fulfillment of the requirements for the degree of

Doctor of Engineering

in

Information Engineering

under supervision of

Yasuhiko Morimoto

Department of Information Engineering

Graduate School of Engineering

Hiroshima University, Japan

September, 2017

Dissertation Summary

Recent computing infrastructure makes a large amount of information available to consumers. Such large volume of data are responsible for information overload problems. A lot of works related to useful information retrieval have been considered to solve the issue. One of the most important primitive function of information retrieval is to select small number of representative objects from a large scale database.

Top-k queries have been extensively used to make a choice of preferable objects from large dataset. A scoring function and a number k, for number of objects to be selected, are specified by users. Then, top-k query returns k objects based on the user defined scoring function. Specifying a scoring function is sufficient and users do not have to define any complex query conditions for retrieving k objects. Therefore, top-k query is more preferable query interface for hand-held devices like smart-phones, tablets and tablets. On the other hand, it is quite possible that scoring functions of every user may not be similar for selecting top-k objects, which indicates that the top-k query results are valuable for those users who share an identical scoring function.

Skyline query, which is also known as a popular information retrieval tool, has been used to eradicate dominated objects. *Skyline* query can be used to select objects that are preferable for all users whose scoring functions are not identical. However, it may retrieve too many or too few objects.

Moreover, in order to retrieve the result of *top-k* or *skyline query*, it is necessary to disclose the values of each data object. In some cases, to compute *top-k* or *skyline query*, we have to disclose sensitive information.

In this dissertation, the author proposed a k-object selection mechanism that chooses various k objects which are preferable for all users who may have non-identical scoring

function; meanwhile, it also ensures the privacy of attribute value during the process of computation.

Now a days, we often have to retrieve necessary objects using hand-held devices like a smart phone or fablets or tablets. In such environment, it is tough to define complex query conditions like scoring function. Users want to retrieve objects by specifying only keywords and the number of objects k. Our proposed method must be useful for such situation.

To achieve above mentioned query, the author used skyline query function. In order to handle so called "big data", The author has considered a distributed algorithm for computing a skyline query.

MapReduce is a popular distributed computing framework for big data applications. To handle large-scale database, proposed algorithm has been developed on *MapReduce* framework. In conventional distributed algorithms for computing a skyline query, the values of each object of a local database have to be disclosed to another. Recently, we have to be aware of privacy in a database, in which such disclosures of privacy information in conventional distributed algorithms are not allowed. In this work, the author has enhanced the security of the distributed algorithm so that the privacy of the data during processing kept intact. In other words, the author proposed a novel approach to compute the skyline in a multi-parties computing environment without disclosing individual values of objects to another party.

The author starts this dissertation from discussion and background of the problem in **Chapter 1**. Then, literature surveys on related topics of the dissertation is presented in **Chapter 2**.

After that, the author splits this dissertation in several parts. The first part of this dissertation considers a novel way of secure skyline computation on MapReduce. The author has considered the situation where owner of dataset are multiparty rather than a single entity. They want to find skyline query result but do not want to disclose any domain

values. Even if parties are not willing to share domain value ranking or order information – as order itself may be considered as sensitive information. The author proposed a novel way to resolve the situation and find the result of skyline query without knowing any domain value. Our proposed algorithm have used MapReduce programming model to ensure the its capability to process big data efficiently. Details of this process in expressed in **Chapter 3**.

In the second part of this dissertation, the author studies the problem of securely k-object selection. When dataset belong to multiparty and privacy of data has become a vital issue, conventional k-object selection algorithms are useless. Our model have addressed the issue and it is capable to find k objects from dataset whose attribute values are not discloseable. The author has discuss the issue in **Chapter 4**.

The third part of this dissertation, the author has considered one of the most important application of the secure k objects selection function, which is the issue of finding key person from social media. Conventional social media mining techniques use graph mining algorithm to mine social media. But, the author has proposed a parallel model to mine social media using skyline query. The author discussed details about this model in **Chapter 5**.

Finally, a concluding discussion with future guideline for extending the work have been discussed in **Chapter 6**.

Contents

List of Figures

List of Tables

The only way to do great work is to love what you do.

Steve Jobs

Acknowledgments

First of all, I would like to express my sincere gratitude to the Almighty for giving me the opportunity to successfully complete my research work. I want to express my sincere gratitude to my supervisor Dr. Yasuhiko Morimoto, Graduate School of Engineering, Hiroshima University, Japan for giving me the opportunity to explore new ideas in the field of knowledge discovery and data mining. Without his moral courage and excellent guidelines, it was not possible for me to complete this work. His advanced knowledge, inspiration and above all diligence expertise in this field provide me many opportunities to learn new things to build my carrier as a researcher. Special thanks are due to the members of this dissertation committee Prof. Kazufumi Kaneda, Prof. Toru Nakanishi and Prof. Jun-ichi Miyao for their invaluable comments and suggestions that enrich my knowledge.

I am thankful to the Japanese Government for providing the Mext scholarship to pursue my study. I am indebted to Rajshahi University(RU), Bangladesh that has granted me a study leave to attend the Doctoral program at Hiroshima University.
Special thanks to all my co-authors and lab members for their support and contributions during my research. Specially I'm grateful to Dr. Md. Anisuzzaman Siddique for his kind support and cooperation since the beginning of my research work. I am also grateful to all of my country mates for their co-operation, support, and warm relationships during my stay in Japan.

Last but not least, I am deeply grateful to my family members for their comprehension, dedication, care, and support throughout the research work.

Chapter 1

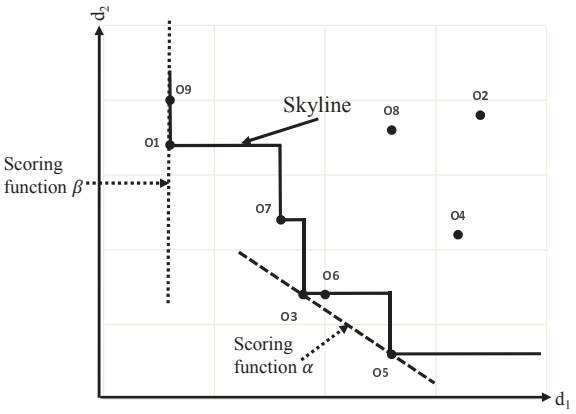
Introduction

At every moment, enormous amounts of data are generated on every domain of life. Such huge amounts of data can be used to interpret people and predict their behaviors more diligently. With the increase of data volume in different aspects of life, advanced query operators are necessary in order to help users to filter the huge amount of available data by selecting a set of promising data objects. Skyline query [9, 22] and its variants [13, 48, 36] are known as such advanced query. Unlike the traditional *SQL* queries, that return a complete result set, skyline queries return all non-dominated objects from a given dataset. An object is said to be non-dominated if it is not worse than any other object in any of the attributes and is better in at least one of the attributes. Perhaps skyline queries have been proposed for filling the gap between set-based *SQL* queries and rank-aware database retrieval [14].

Let us consider the example of Figure 1.1. In the given example, we can see that Figure 1.1(a) consists of a two dimensional dataset: d_1 and d_2 – along with data IDs. If we compare the data object O_4 with another data object O_2, we can see that O_4 is better than O_2 in every dimension (assuming smaller value is better). In such case, we say, O_4 is dominating O_2 or O_2 is dominated by O_4. However, it is also understandable that O_4

id	d_1	d_2
O_1	03	17
O_2	17	19
O_3	09	07
O_4	16	11
O_5	13	03
O_6	10	07
O_7	08	12
O_8	13	18
O_9	03	20

a) Symbolic Dataset



b) Skyline & top-k query

Figure 1.1: Understanding Skyline and top-k query

itself is dominated by O_5. While comparing O_5 with O_6, none can be claimed to be better in every dimensions. Hence, they do not dominate each other. The author discussed the mathematical definition of dominance and skyline query in Chapter 2.

As the author mentioned that skyline query results are composed by the non dominated objects, we can find from the Figure 1.1(b) that $\{O_2, O_4, O_6, O_8, O_9\}$ can not be member of skyline query result set − they are somehow dominated by some other objects within the dataset. However, there exists no other data objects within the given data set that can dominate $\{O_1, O_3, O_5, O_7\}$. Hence, they are the result set of skyline query.

Top-k query is also known to be a popular tool in finding top k elements from a dataset. In top-k query user must specify some scoring functions. Based on scoring functions top k elements are ranked and returned to the user. Let us again consider the example given in Figure 1.1(a) where a symbolic dataset is defined with d_1 and d_2 attributes. Users may evaluate data points with his/her own scoring function. If a user's scoring function defined by the lowest aggregate of d_1 and d_2, which we call "*scoring function* α" as in Figure 1.1(b), then top-2 data objects based on "*scoring function* α" are O_3 and O_5. However, another user may look for the lowest d_1 and have no interest in d_2, which we can be evaluated by "*scoring function* β" as in Figure 1.1(b). Top-2 answer based on "*scoring function* β" are

O_1 and O_9. However, it is to be mentioned that, to specify scoring function user must have some domain knowledge. Perhaps, without having domain knowledge user can not specify scoring function correctly. Moreover, scoring function of all user are not unique. Hence, top-k result will vary from user to user.

Social networks or social media are computer based technology that allows us to create and share information, events, interests, photos and many other forms of expression thorough virtual communities or networks. It has vast amount of services, some are standalone others are built-in. It is hard to define social media in some few sentences. However, according to researchers, social medias have some common features like they are usually Web 2.0 based interactive applications, contents (usually texts, photos, events, etc.) are generated by users. User creates service-specific profiles which are managed by the service provider and system facilitates users to be grouped together by connecting user profiles. Gigantic amount of data are generated in every moment in social media. These dataset are huge and complex. Big data is a term for dataset that are too large or complex that traditional data processing application software is inadequate to deal with them. In order to process big data, conventional sequential processing algorithms and techniques are not efficient enough. Parallel algorithms are often required to process such complex data. While designing parallel algorithms, designers must consider new issues like robustness, fault-tolerance etc, − as well as computational efficiency retirements. To aid designers and programmers, different programming models have been introduced in last few decades. MapReduce, proposed by Google Inc., is one of such parallel programming models. This programming model was designed to process huge amount of data (i.e. big data) with a parallel, distributed algorithms on a cluster. While using such robust framework, programmers do not have to worry about factors like redundancy, fault tolerance etc. A MapReduce framework orchestrates the processing by controlling the distributed nodes, running various tasks in parallel, managing all communications and data transfers between the various

parts of the system, and providing for redundancy and fault tolerance. Hadoop, maintained by Apache Software Foundation, is an open source implementation of Google's MapReduce framework.

1.1 Motivation

Privacy of data has become a burning issue in database community for last few decades. Information privacy or data privacy is known to be the relationship between collection and diffusion of data, technology, the privacy expectation of mass people, and the political & legal issues surrounding them. Privacy issues are important whenever sensitive or personally identifiable data are collected, stored, processed and destroyed in any forms − especially in digital form. Sometimes, we have to hide individual values to preserve privacy, and even we may not be allowed to disclose aggregate values or order values of data objects. We know that in computation of skyline query result it is necessary to compare domain (or attribute) values of each data objects. Even if for any efficient or tricky algorithms, like BBS(*Branch-and-Bound Skyline*) [32] or *Sort-Filter-Skyline*(SFS) [12], it is essential to know the domain values prior computation. Lets consider the situation when the ownership of data does belong to several persons or organizations, who are not interested to disclose the domain values but want to calculate the result of skyline query. Conventional skyline query algorithms are unable to resolve such situations. As a part of this research work, the author proposed a novel approach to address the issue in distributed environment. The author proposed a model to retrieve skyline query result securely on MapReduce framework.

k-object selection problem was introduced in [37]. It is an interesting query for the user who wants to find top k objects but do not have any domain knowledge. Conventional top-k query requires some scoring function to identify the top k elements from the dataset. However in k-object selection problem, the authors proposed a novel algorithm to recommend

k objects for those users who do not have any prior domain knowledge. The algorithm was designed by using skyline query. In previous model [37], it was assumed that domain values of data objects are disclosable. But in the situation when data objects are property of different entities and corresponding entities are not willing to share its data, that algorithm will not be useful. The author enhanced the algorithm and proposed a secured model to solve the k-object selection problem. To overcome computational bottleneck imposed by single core computation, the author designed this algorithm using MapReduce paradigm.

Social media like Facebook, tweeter or LinkedIn are good source of valuable information. These medias generate enormous amount of data. The rate of growth of data volume is increasing day by day. Such huge amount of data can be used to understand mass people more effectively. It is also possible to use these huge data to interpolate mass peoples' choice or their way of thinking. One way to do it is to find some key persons from social media. These key persons may be used as ambassador of some certain product or opinion. Most of the social mining techniques are designed as graph mining problem. The author proposed a model, using Google's MapReduce framework, where the author used skyline query to find some key persons from social media. The author started with designing a model where key persons of social media can be identified using three feature matrices and skyline query's dominance feature. Later the author upgraded our model using five feature matrices. Our final model uses five social matrices: Friend power(F_p), Followee strength (F_s),"Like" score (L_s), Comment support (C_s), and Group score (G_s) along with dominance power of skyline query.

1.2 Thesis Organization

The rest of this thesis is organized as follows: First of all the author discussed some fundamental knowledge and related works that are required to understand this work in **Chapter 2**. **Chapter 3** discuss details about proposed model where the author introduced

a novel way to compute skyline query result without disclosing domain values. The author designed the model base upon the distributed computing framework − MapReduce. In **Chapter 4** the author reviewed the model of secure k-object selection problem. This model was also designed based on the Google's MapReduce framework. In **Chapter 5** the author discussed about the mining problem of social media. This chapter discuss about our model of selecting or finding key persons of social media. This model was proposed using skyline query for selecting key persons of social media. Finally, **Chapter 6** concludes our work and provide some future guideline to go forward.

Chapter 2

Background Knowledge

In this chapter, the author discussed fundamental knowledge and related works in details.

2.1 Skyline Query

In database researchers community, skyline query is known to be one of the most popular and useful query tool for last few decades. A skyline query retrieves a set of objects, each of which is not dominated by another data objects. Consider the example of a two dimensional database, given in Figure 2.1. As we know, skyline query retrieves all non dominated objects, we can see that there exists no other objects which can claim to be better than $\{O_2, O_4, O_7\}$. Hence they are the result of skyline query. On the other hand, there exists at least one objects who can claim to be better than each of the remaining objects: $\{O_1, O_3, O_5, O_6\}$. That means they are dominated by some one else. That's why they are eliminated from the result set.

In order to define dominance relationship and skyline query mathematically, let us assume that there is an n-dimensional dataset DS, $\{d_1, d_2, \ldots d_n\}$ be the n attributes of DS and, m objects $\{O_1, O_2, \cdots, O_m\}$. This dissertation uses $O_i.d_j$ to denote the j-th dimension value of object O_i. Without losing generality, we assume that smaller value in

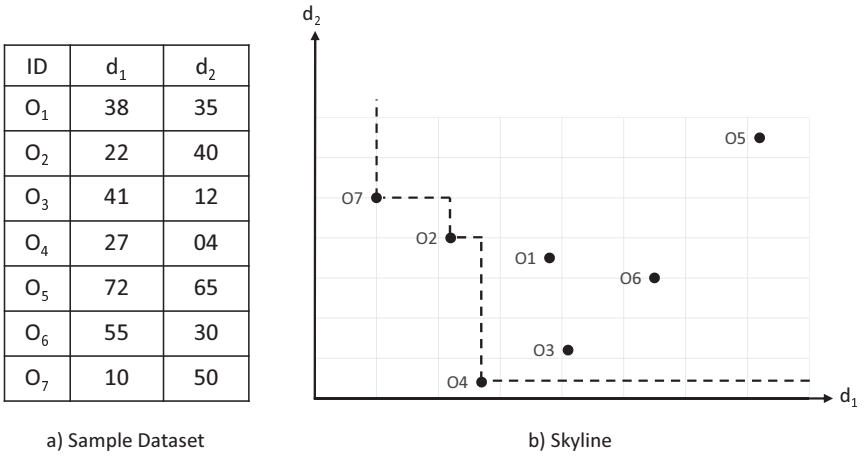


Figure 2.1: Skyline query

each attribute is better.

Dominance: An object $O_i \in DS$ is said to dominate another object $O_j \in DS$, denoted as $O_i \prec O_j$, if $O_i.d_r \leq O_j.d_r$ $(1 \leq r \leq d)$ for all d dimensions and $O_i.d_t < O_j.d_t$ $(1 \leq t \leq d)$ for at least one attribute. We call such O_i as *dominant object* and such O_j as *dominated object* between O_i and O_j. Example given in Figure 2.1, O_3 is better than O_6 in every dimension, hence $O_3 \prec O_6$ but in between O_3 & O_1, $O_3 \not\prec O_1$ and $O_1 \not\prec O_3$.

Skyline: An object $O_i \in DS$ is said to be a *skyline object* of DS, if and only if there is no such object $O_j \in DS$ $(j \neq i)$ that dominates O_i. The skyline of DS, denoted by $Sky(DS)$, is the set of skyline objects in DS. For example in Figure 2.1, $Sky(DS) = \{O_2, O_4, O_7\}$.

Candidate Skyline: Candidate skyline is a super set of *skyline*, which means that the candidate skyline must include all skyline objects though it may include non-skyline objects. We use candidate skyline because it can be computed efficiently in distributed environment and it is effective to prune dominated objects.

Candidate skyline is a set of objects that can be selected as follows. (1) We make a list of sorted objects for each attributes. For example, sample dataset given in Figure 2.1, we make two lists: $\{O_7, O_2, O_4, O_1, O_3, O_6, O_5\}$ on d_1, $\{O_4, O_3, O_6, O_1, O_2, O_7, O_5\}$ on d_2. (2) We iteratively pop the top object from the sorted lists and add the object into candidate

skyline if the object is not in the candidate skyline. If the object is already in candidate skyline, we increment counter of the object, which keeps how many time the object is poped from the sorted lists. We continue the iterative procedure while all the counter values are less than the number of attributes. In the example, $\{O_7, O_4\}$ are added into candidate skyline. Similarly, we add $\{O_2, O_3\}$. Then, we can find that the next object O_4 is already in candidate skyline and the counter for O_4 becomes 2, which is the number of attributes. Therefore, $\{O_7, O_4, O_2, O_3\}$ are *Candidate skyline* in Figure 2.1. Note that if one of the counter values becomes the number of attributes we can prune other objects that are not in candidate skyline since all the other objects are dominated by the object whose counter value is the number of attributes. In the example, O_4 dominates all other remaining objects.

It is possible to calculate *Candidate skyline* list without doing any single dominance test. As the complexity of *skyline* computation depends on dominance test, it may be considered as an efficient way of pruning *non-dominant* data objects. The way of calculating *Candidate skyline* is explained in one of our paper [38].

2.2 Secure Skyline Query

The idea of "Secure skyline query" comes from the necessity of data privacy issue. Now a days, computational complexity as well as data privacy have attracted considerable importance from researcher community. From the definition, we know that: in order to find the result of skyline query we have to check dominance relation. Dominance relation can not be verified without knowing domain values of each attribute. Tricky algorithms like *Branch-and-Bound Skyline*(BBS) [32] or *Sort-Filter-Skyline*(SFS) [12] also require domain values to be disclosed. Assume a situation where the dataset belongs to different parties and they are willing to find the result of skyline query but do not want to disclose their domain values. To understand the query problem, lets consider the example

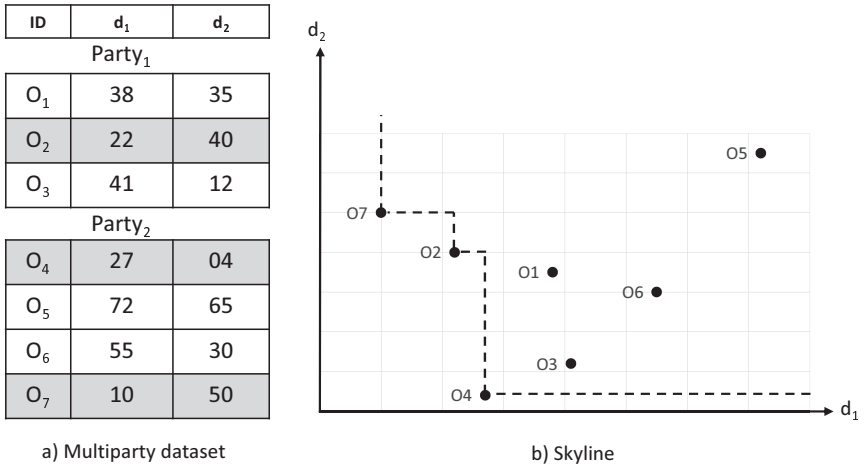


Figure 2.2: Secured Skyline query Problem

given by Figure 2.2. As shown in that example, data objects $\{O_1, O_2, O_3\}$ belongs to $Party_1$ and $\{O_4, O_5, O_6, O_7\}$ belongs to $Party_2$. In case where $Party_1$ & $Party_2$ want to find the skyline query result but do not want to share domain values, skyline computation using conventional technique become impossible. Using conventional technique we have to perform dominance check between objects of $Party_1 : \{O_1, O_2, O_3\}$ and objects of $Party_2 : \{O_4, O_5, O_6, O_7\}$ without knowing their d_1 & d_2 values – which is technically impossible. The proposed model, explained in Chapter 3, addressed the issue and found some solutions of this problem. The solutions were designed by using Google's *MapReduce* programming models and implemented on *hadoop* framework.

2.2.1 Hadoop - MapReduce

MapReduce is a programming model and software framework first proposed by Google Inc. This programming model was designed to process huge amount of data (i.e. big data) with a parallel, distributed algorithms on a cluster. While using such robust framework, programmers do not have to worry about factors like redundancy, fault tolerance etc. A *MapReduce* framework orchestrates the processing by controlling the distributed nodes, running various tasks in parallel, managing all communications and data transfers between

the various parts of the system, and providing for redundancy and fault tolerance.

Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model.

Hadoop is an open source implementation of the *MapReduce* framework, maintained by Apache Software Foundation [5]. This framework is designed to allow users to define a *MapReduce* job only by defining the *map* and *reduce* functions. In this framework, data are represented as $\langle key, value \rangle$ pairs and computations are distributed across a shared nothing cluster of autonomous machines. Jobs to be performed using the *MapReduce* framework mainly refer to two user-defined functions, called *Map* and *Reduce*:

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

$$Reduce(k_2, list(v_2)) \rightarrow list(v_3)$$

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 intermediate *key*. The *Reduce* function (sometimes called Reducer) takes a *key* and a list of *values* for that *key*, applies the processing algorithm, and generates the final result. Models proposed by this research work are basically *MapReduce* based algorithms. We designed algorithms which explore the strength or power of parallel computing in distributed environment.

2.2.2 Secure Skyline Query Related Works

Proposed "Secure Skyline Query" model is motivated by previous studies of skyline query processing, multiparty secure computation, as well as *MapReduce* based query processing.

Skyline Query

Borzsonyi et al. have proposed the skyline operator over large databases and proposed three algorithms, which are *Block-Nested-Loops*(BNL), *Divide-and-Conquer* ($D\&C$), and B-tree-based schemes [9]. BNL compares each object of the database with every other objects, and reports it as a result only if any other object does not dominate it. $D\&C$ divides the dataset into several partitions such that each partition can fit into memory. Local skyline objects for each individual partition are then computed by a main-memory skyline algorithm. The final skyline is obtained by merging the local skyline objects for each partition. Kossmann et al. improved the $D\&C$ algorithm and proposed nearest neighbor (NN) algorithm for efficiently pruning dominated objects by partitioning the data space iteratively based on the nearest objects in the space [22]. Chomicki et al. proposed *Sort-Filter-Skyline*(SFS) as a variant of BNL [12], which can improve BNL by presorting. The most efficient method so far is *Branch-and-Bound Skyline*(BBS), proposed by Papadias et al., which is a progressive algorithm based on the *best-first nearest neighbor (BF-NN)* algorithm [32].

Recently, parallel computing paradigm becomes very popular for skyline computation. W.T. Balke et al. have introduced skyline queries in distributed environments in [7]. Vertically partitioned web information are supported by their work. Thereafter, within the literature, abundant studies achievements had been received to address distributed skyline queries. Both of Wang et al. and Chen et al. researched skyline queries in structured P2P networks, named BATON networks, where peers are responsible for a partial region of data space [44]. Rocha-Junior et al. [35] proposed a grid-based approach for distributed skyline processing (AGiDS), which assumes that each peer maintains a grid-based data summary structure for describing its data distribution. Arefin et al. [6] worked on agent based privacy skyline-set for distributed database but the problem solved in this paper is different from the conventional skyline query, which we are considering in this work.

The *MapReduce* framework, which have been developed by Google, has become popular to process queries over big data due to its scalability and fault tolerance. In [48], Zhang et al. first proposed a preliminary way for skyline queries in *MapReduce* framework. Three skyline algorithms, called MR-BNL, MR-SFS and MR-Bitmap, were proposed by the author using the *MapReduce* framework. In [33], Park et al. introduced an efficient parallel algorithm, SKY-MR, for processing the skyline queries. In the SKY-MR algorithm, a sky-quad tree was introduced with a sample of the entire dataset and was utilized in the data partition and local pruning.

Multi-party Secure Computation

Multi-party computation problem, which was introduced by Yao [46] and extended by Goldreich, Micali, Wigderson [17] and many others, has attracted attention in privacy-aware computing environment. Secure function evaluation, as was introduced by Yao, allows a set $P = \{p_1, \cdots, p_n\}$ of n players/parties to compute an arbitrary agreed function of their private data, even if an adversary may corrupt and control some players/parties in various ways − the privacy of data will be preserved. Security in Multi-party Computation means that the parties' data remain secret (except for what is revealed by the intended results of the computation) and that the results of computation are guaranteed to be correct[17]. In general, Multi-party Computation protocols tend to be less efficient than specific purpose protocols.

In privacy-preserving data mining problems, there are another multi-party secure computation problems that have been discussed in the literature. Lindell and Agrawal proposed two different privacy preserving data mining problems. In the problem defined in Lindell's paper [26], two parties, both having non public databases, want to jointly conduct a data mining operation on the union of their two databases. The problem is how to compute the operation without disclosing their database to other parties, or any third party. On

the other hand, in Agrawal's paper [3], the problem is as follows: one party say "Alice" is allowed to conduct data mining operation on a private database owned by another party say "Bob", the problem is how could Bob prevent Alice from accessing precise information in individual data records, while Alice is still able to conduct the data mining operations. Lindell and Pinkas used secure multi-party computation protocols to solve their problem, while Agrawal used the data perturbation method.

MapReduce Implementations of Skyline Query

Google's *MapReduce* [8, 19, 42] and its open source variant *Hadoop* [5] are powerful tools for building scalable parallel applications. Recently, *MapReduce* has attracted a lot of attention in handling "big data". There exist some recent works on large-scale skyline computation using *MapReduce* [21, 40]. In [36, 41], we have proposed a *MapReduce* based algorithm to process k-dominant skyline query. The k-dominant skyline query can reduce the number of retrieved objects by relaxing the dominance definition, in which an object more likely to be dominated. However, all of the above methods cannot preserve privacy of individual object during the distributed computation.

2.3 k-object Selection Query

k-object selection query is inspired by top-k query. The top-k query is considered as another popular query tool in database researchers community. In order to retrieve results, in top-k query, user have to define some scoring or ranking functions. Based on the scoring function top k elements are returned to the user. But in situation when users do not have any prior domain knowledge, it is difficult to specify scoring function. Moreover, scoring function of different users may vary a lot. Hence, top-k results for one user may useless for another user. To resolved the issue, k-object selection query was introduced in [37]. The result of k-object selection query ensures that the outcome of the query will be useful to every user

who have different linear scoring function on any domain attribute. Moreover, in k-object selection query user do not have to specify any scoring function, hence, necessity of domain knowledge is eliminated.

Top-k Query: If k be a positive integer and $\mathbf{w}$ be a weighting vector then the result set $TOP_k(\mathbf{w})$ of the top-k query is a set of objects such that $TOP_k(\mathbf{w}) \in DS$, $|TOP_k(\mathbf{w})| = k$ and $\forall O_i, O_j$: $O_i \in TOP_k(\mathbf{w})$, $O_j \in DB - TOP_k(\mathbf{w})$ it holds that $F_{\mathbf{w}}(O_i) \leq F_{\mathbf{w}}(O_j)$ where $F_{\mathbf{w}}(O)$ is the inner product value of $\mathbf{w}$ and O. In top-k query the number of retrieved objects can be specified. However, specifying weight vector is not an easy procedure for users. The information-retrieval society have been using keyword-based querying for decades. In such information retrieval system a user specifies a keyword and gets necessary objects that are closely related to that keyword. We assume that a user can specify k,the number of necessary object, but can not specify any weighting vector.

2.3.1 k-object Selection Query Related works

Top-k about skyline queries were studied in [25, 32, 39]. Ranked skyline and k-dominating queries were introduced by Papadias, *et al.* in [32]. If a set of objects in d-dimensional space are given, a monotonic ranking function returns k objects in the d-dimensional space which have the best (smallest or greatest) scores according to an input function. However, k-dominating queries retrieve k objects that dominate the greatest number of objects. Representative skyline query is discussed by Lin *et al.* in [25]. The query is formulated as to select k skyline objects according to a objective function. The objective function is pre-defined. Representative skyline query finds a set of k objects among all skyline objects such that the number of objects dominated by this set is maximized.

An alternative definition of representative skyline query was defined by Tao, *et al.* [39]. A representative skyline query finds k representative objects from all skyline objects such that the sum of the distances between each skyline object and its "closest" representative

object is minimized. All of the idea mentioned above are to find k objects given a dataset DS where attribute preferences or scoring functions in the dataset are also given.

2.4 Social Network Mining

Social network, also known as social media, are computer based methodology that allows us to create and share information, events, interests, photos and many other forms of expression thorough virtual communities or networks. Due to the vast amount of social media services (standalone as well as built-in), it is hard to define social media in some few sentences. However, according to researchers opinion, social medias have some common features: they are usually Web 2.0 based applications, contents (text, photo, events, etc.) are generated by users, user creates service-specific profiles - managed by the service provider and system facilitates users to be grouped together by connecting user profiles, etc. Gigantic amount of data are generated in every moment in social media. These huge amount of data can be used to manipulate mass people opinion more efficiently. Finding some key persons could have been an interesting use of such huge amount of data. Most of the researchers have considered social media mining as a problem of graph mining and shown efficient ways to use graph mining algorithms. In our proposed model we have used the dominance check principal of skyline query to find some key persons from social media. We have used five social media matrices to address the issue and our proposed model uses Google's *MapReduce* frame work architecture in order to achieve parallelism to process enormous amount of data from social media.

2.4.1 Social Media Mining Related Works

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 [16]. 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 [23]. Cao et al. have addressed the jury selection problem by using micro-blog services like Twitter to solve decision-making tasks [10]. 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 [4] [28], 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 research work complements the existing efforts for addressing key the person selection problem and creates an opportunity to recruit them as brand ambassador.

Chapter 3

Secure Skyline Query

A skyline query retrieves a set of representative objects, each of which is not dominated by another object.

Recently, we have to be aware of privacy of individual object in a database. So far, a lot of algorithms for computing skyline query have been proposed, some of which are designed in distributed computing environment to be able to handle "big data" [1, 21, 33]. However, none of them considered privacy issues in a database. In this Chanter, the author proposed a novel approach so that data can be processed in distributed manner, meanwhile privacy of individual object has been preserved.

Assume that many organizations have done some surveys about commission cost and risk prediction. Assume that each of the organizations has collected a same kind of privacy information of their customers. Since each organization does not want to disclose the database, each can not compute skyline query of the union of all organizations' databases but only compute skyline query of its own database. It is no doubt that the skyline of the union is more valuable than that of each.

Suppose, for example, two individual organization have done some market research and they have collected a database about commission cost and risk prediction. These

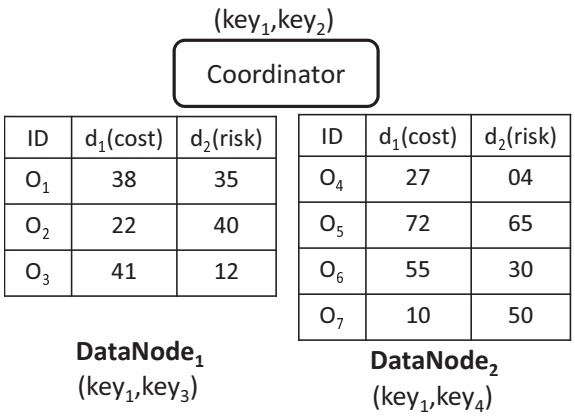


Figure 3.1: Multi-Party *Skyline* Computation

information are sensitive and both the parties are not wanting to disclose the values of individual object. But, these two parties are willing to get the outcome of skyline query of the cost and the risk from the union of the two parties' database. In conventional skyline computation methods, it is not possible to get the result of skyline query result without disclosing the values to others. Proposed method can solve this issue, not only by preserving data privacy but also by doing the job in distributed *MapReduce* framework.

The remaining part of this chapter is organized as follows.

Section 3.1 discusses about the problem statement. In Section 3.2, the author specified details of the algorithms with proper examples and analysis.

3.1 Multi-Party Secure Skyline Problem

Let us consider a situation where several organizations have done some surveys about commission cost and risk prediction. This paper assumes that each of the organizations has collected a same kind of privacy information of their customers. Each organization wants to find the result of skyline query of the union of these organizations' databases. But, none of them is allowed to disclose values of their database to other organizations.

We call participant organizations of the skyline computation as parties. To accom-

plish the computation, we use a trusted third party as *Coordinator*. We assume that *Coordinator* is not vulnerable to intruder and will not disclose sensitive information.

To simplify the problem, we assume the number of participant parties is two and denote the two parties as $DataNode_1$ and $DataNode_2$, respectively.

In order to ensure the privacy of communication between the *Coordinator* and *DataNode*s, the *Coordinator* uses *Public-key* cryptography, that is the *Coordinator* is equipped with a public key (key_1) and a private key (key_2). *DataNode*s are informed about the key_1 of *Coordinator* and before sending any data packet to *Coordinator*, it must be encrypted by key_1. Only *Coordinator* can decrypt the data using key_2. Similarly, both *DataNode*s use the *Symmetric-key cryptography* for their own data's privacy. Let the cryptography keys be key_3 and key_4 for $DataNode_1$ and $DataNode_2$, respectively.

3.2 Multi-Party Secure Skyline Algorithm

The proposed algorithm consists of six steps: (1) Preparing the $\langle key, value \rangle$ pair, (2) Ordering with *MapReduce*, (3) Disguise the original order, (4) Return of disguised order values, (5) Merging and sorting, and (6) Skyline computation.

3.2.1 Preparing the $\langle key, value \rangle$ pair

In order to avoid confusion, we will denote *encryption-key* for the cryptographic key to distinguish with the *key* of $\langle key, value \rangle$ pairs, which are used in MapReduce framework. In Figure 3.1, each *DataNode* has a collection of two dimensional sensitive data. Both *DataNode*s perform *Symmetric-key cryptography* (e.g. DES -Data Encryption Standard) with private *encryption-key* before sharing any data with *Coordinator*.

Figure 3.2 illustrates the encryption and how to generate $\langle key, value \rangle$ pair's of the first digit, $digit_1$, (the least significant digit). In the example, only the dimension d_1 is described though the same encryption were performed on all the sensitive dimensions. Each party

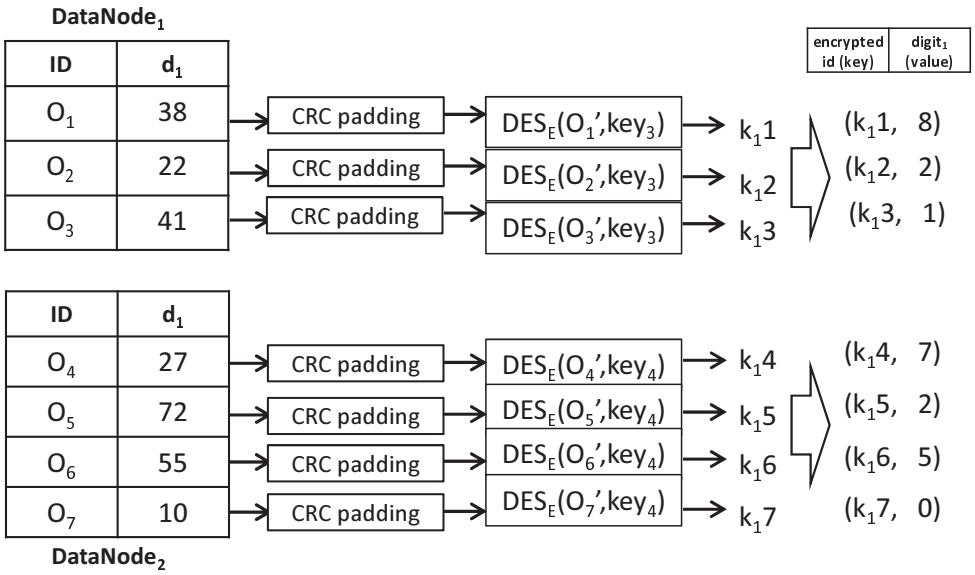


Figure 3.2: Generation of $\langle key, value \rangle$ pair

encrypts their data as follows: (1) Each $DataNode$ adds redundant bits for each ID by using CRC(*Cyclic Redundancy Check*) scheme [45]. We call this process as "CRC Padding". Assume that, for the example case given in Figure 3.2, first record of $DataNode_1$ whose ID is O_1, after padding modulo string becomes O_1'. (2) IDs with padded CRC bits, are encrypted by the corresponding node's encryption key. In example, we generate "*key*" ($k_1 1$) for the first record of $DataNode_1$ (whose original ID was O_1 , after padding it became O_1') by encrypting with key_3. Note that owner of the *encrypted_id* can decrypt but any other parties cannot. Though CRC scheme is usually used as a transmission error checking tool, we also used the CRC scheme to check whether a processed $\langle key, value \rangle$ is $DataNode$'s (is inherent in $DataNode$). (3) We generate $\langle key, value \rangle$ pairs of the first digit (the least significant digit). For example, $\langle key, value \rangle$ pair of the first record of $DataNode_1$ is $\langle k_1 1, 8 \rangle$ (as the $digit_1$ of dimension d_1 for object O_1 is 8) .

In our method, a key (an *encrypted_id*) of a $\langle key, value \rangle$ pair must be unique. However, there is a possibility that different records happen to have a same key (*encrypted_id*) though the possibility is extremely small. If such a collision is found by $Coordinator$, $Coordinator$

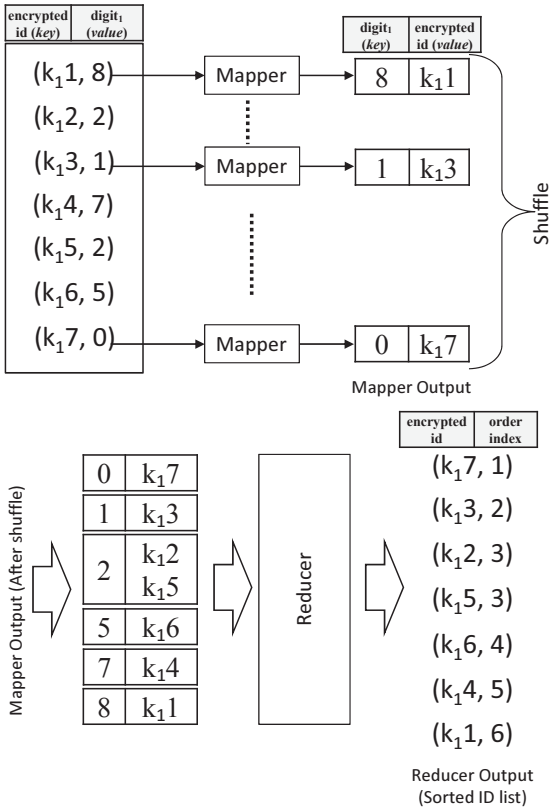


Figure 3.3: Ordering with *MapReduce*

forces all *DataNode*s to reproduce keys (*encrypted_id*s) with different "CRC Padding". Hence the possibility of conflicting in "CRC Padding" can be eliminated.

3.2.2 Ordering with *MapReduce*

In this stage, we order the *encrypted_id*s based on their $digit_1$ values in distributed manner with *MapReduce* framework. The encrypted values of data IDs and corresponding $digit_1$ values are stored in distributed file system (*DFS*) more specifically in *HDFS* (*Hadoop Distributed File System*). The *Mapper* reads each $\langle encrypted\_id, digit_1 \rangle$ and depicts as *Mapper*'s output: $\langle digit_1, encrypted\_id \rangle$. According to the working principle of *MapReduce* framework, the $digit_1$ serves as *key*. *MapReduce* framework will shuffle and sort the $\langle key, value \rangle$ pairs so that the *key* ($digit_1$) values are ordered and tagged together. The *Reducer* layer collects the shuffled values and produce the sorted order of *encrypted_id*s.

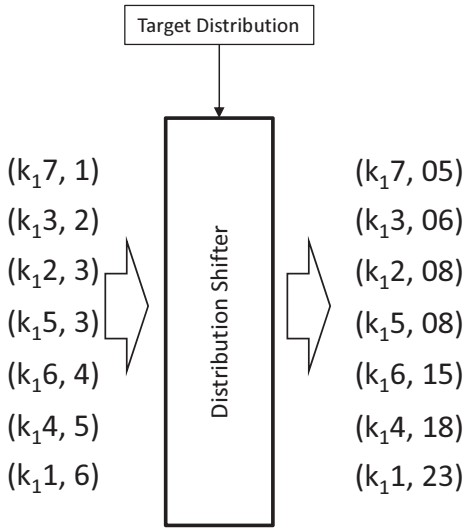


Figure 3.4: Disguise the original order

Note that several *encrypted_id*s may have same $digit_1$ value as *key* and they will have the same ranking index.

In Figure 3.3, we can see that a *Mapper* takes the $\langle k_1 1, 8\rangle$ pair as input and produces $\langle 8, k_1 1\rangle$ as output. Similarly, a *Mapper* takes $\langle k_1 3, 1\rangle$ and produces $\langle 1, k_1 3\rangle$, and so on. After the shuffling, pairs are grouped by *key* and are feeded into *Reducer* layer. *Reducer* produces the sorted order of *encrypted_id*.

3.2.3 Disguise the original order

In order to conceal the actual order ({1, 2, 3, ...}) from intruders, we had better to shift the density of the order distribution. The detailed idea of the order value transformation is discussed in [2]. Brief explanation of the transformation is as follows: First of all, we have to select a target distribution other than uniform distribution. Target distribution is a user specified data distribution function such as Gaussain or Zipf or similar distribution. After choosing the target distribution, we have to generate $|X|$ unique values from the target distribution where X is the collection of ordered sequence indices. Then, we sort the generated random values into a table T. The sorted i^{th} index value of ordered list

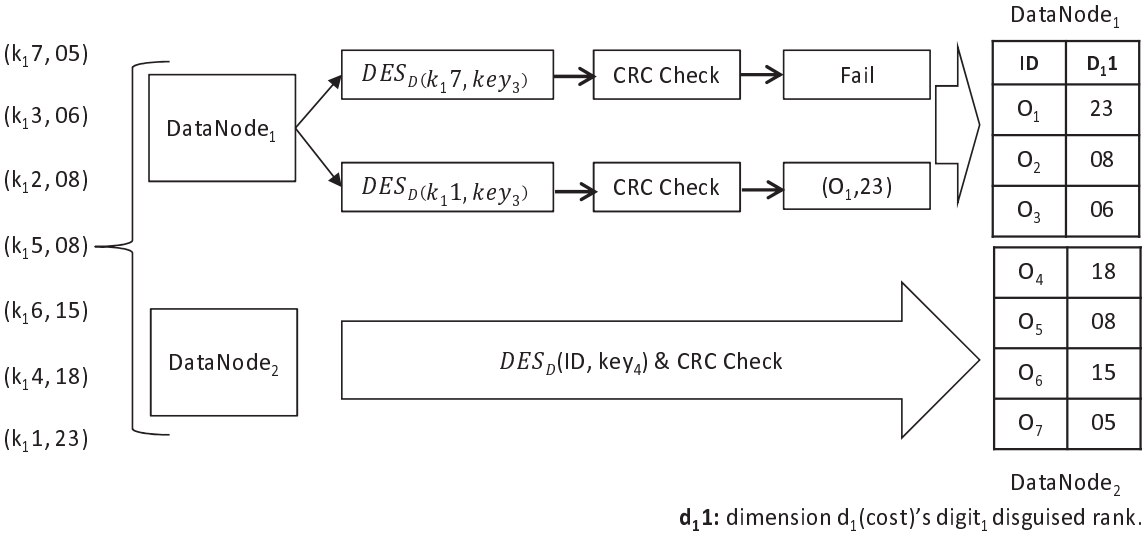


Figure 3.5: Decryption and gathering order

(*encrypted_id*) is then given the order value of *T[i]*.

In Figure 3.4, our *Distribution Shifter* module receives the target distribution and generate 6 unique (because, $|X|= 6$) values and after ordering $T=$ {*05, 06, 08, 15, 18, 23*}. From previous step, we know that $k_1$7's rank is 1 in our sorted list. We will replace this rank with the first value of T, i.e., the new order index of $k_1$7 is 05. Similarly, $k_1$3's order index will be 06 and so on. Hence, we have been able to shift the order index while preserving the order sequence − which we call as disguised order.

3.2.4 Return of disguised order values

Disguising the rank by the previous procedure makes the order of *encrypted_id*s difficult to find while it preserves the original order. After transferring distribution of the rank values, the results are sent back to both *DataNode*s. Note that other parties cannot infer the sensitive information since the original IDs are encrypted.

After receiving the result of the previous step, each *DataNode* tries to decrypt the *encrypted_id* with its own *encryption-key*. If the *DataNode* owns the received data, the *DataNode* can easily identify it by the decrypted ID and the CRC code checking. For

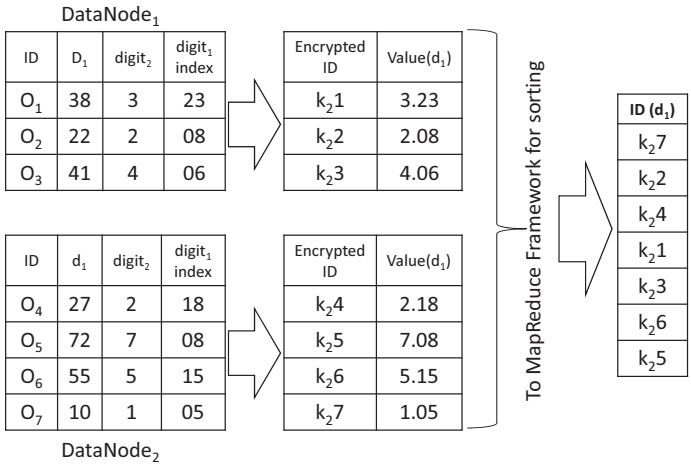


Figure 3.6: Merge and Sort

example, in Figure 3.5, when $DataNode_1$ tries to decrepit k_17 with its encryption key, key_3, it will get some value. Since the k_17 does not belong to $DataNode_1$, the CRC check "Fails". As the CRC "Fails", $DataNode_1$ discards further process of k_17 data segment. On the other hand, k_11 passes the CRC check of $DataNode_1$. Then, $DataNode_1$ processes k_11 for the next round. Similar process is followed by each $DataNode$.

3.2.5 Merging and Sorting

After receiving the disguised order of $digit_1$, each $DataNode$'s next job is to merge those values with $digit_2$ values. These new values along with encrypted IDs are sent to *MapReduce* framework for sorting to get the order of $digit_2$ and $digit_1$. The encryption of IDs can be done, after CRC padding, by using the *encryption-key* as mentioned in Section 3.2.1. The *encryption-key* may be different to that used in $digit_1$. In Figure 3.6 show the process of the $digit_2$ values. In $DataNode_1$, O_1 has been encrypted to a new *encrypted_id* "k_21". Since $digit_1$'s disguised order of O_1 is 23 and $digit_2$'s value is 3, $DataNode_1$ constructs the concatenated value 3.23, which makes the order of $digit_1$ preserved. Similarly, for O_2, the concatenated value becomes 2.08. After calculating all the concatenated values of dimension d_1, we feed these data to *MapReduce* framework to get the order of ID's. Similarly,

we calculate the order of other dimension's values. These orders will be used to calculate a skyline query.

3.2.6 Skyline computation

From the output of *MapReduce* in the previous step, the *Coordinator* get the order of data on each dimension. *Coordinator* uses these order to calculate candidates of skyline query. The detailed process of this skyline calculation, which does not use individual values but the orders, has originally published in [38]. In the work [38], the authors partition dataset vertically and sort each partition. Then, Coordinator collects (ID, Rank) pairs. Figure 3.7 show the example. The left table of the figure has the *encrypted-id*s ordered by each dimension rank value. $ID(d_1)$ is the sorted ID of d_1, which shows that object k_27 is the 1st and object k_22 is the 2nd and so forth. Similarly, object k_24 is the 1st on d_2 and object k_23 is the 2nd and so forth.

Coordinator maintains a counter for each object. In the beginning, the counter for each object is set to zero. *Coordinator* reads the *encrypted-id*s from the 1st row and increments the counter values based on the row value. In the running example, the counter of k_27 and k_24 are incremented by the 1st row. Next, the counter of k_24 and k_23 are incremented by the 2nd row. Next, the counter of k_24 and k_25 are incremented by the 3rd row. If one of the counter becomes the number of dimensions, *Coordinator* will stop the increment procedure. In the running example, *Coordinator* stops the increment procedure when k_24's counter becomes 2 in the 3rd row. Then, *Coordinator* collects IDs whose counter value is not 0 as a candidate skyline. In the example, the candidate will be $\{k_27, k_22, k_24, k_23\}$. Note that other objects must be dominated by k_24 whose counter values is 2.

After the candidates are collected, we examine dominance for all pairs of the candidate objects. We use a sort filtering skyline (SFS) method [12] based on the order of the counter values. As the number of candidates are much smaller than the number of objects

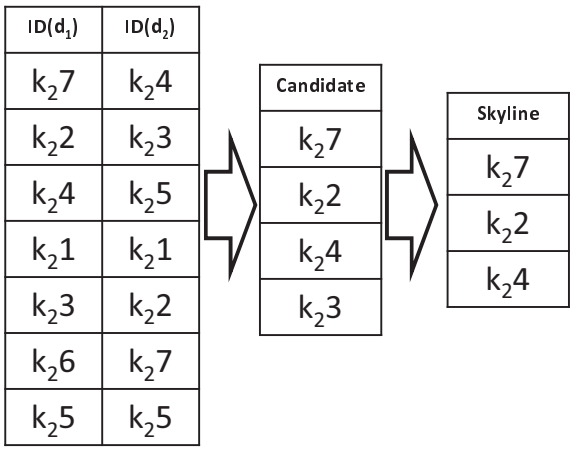


Figure 3.7: Candidate Generation and skyline Computation

any skyline processing algorithm can check the dominance without spending a lot of time. In the example, we can find that $k_2 3$ is dominated by $k_2 4$ and *Coordinator* finalizes the skyline result as $\{k_2 7, k_2 2, k_2 4\}$ which are *encrypted-id*s of $\{O_7, O_2, O_4\}$, respectively.

In Figure 3.7, the left table, which has order for each dimension, contains *encrypted-id*s. In the figure, we explain the candidates generation procedure assuming $k_2 1$ in d_1 and $k_2 1$ in d_2 is the same. But, we have to note that the *encrypted-id*s for d_1 and those of d_2 are different, which prevents compromising relative merits of two anonymous records. In the candidate generation process, each *DataNode* discloses the identity of *encrypted-id*s to *Coordinator*. Only *Coordinator* knows the information of the left table. Each *DataNode* does not find the information of records in the table that are owned by others.

3.3 Experiments

This section reports our experimental results to examine the effectiveness and efficiency of proposed method. We have configured a cluster of 4 commodity PCs in a high speed Giga-bit Ethernet networks, each of which has an Intel Core 2 Duo E8500 3.16 GHz CPU, 8GB memory. We have compiled the source codes under *Java* V8. We have used *Hadoop* version 2.5.2 and 64 bit Cent-OS 7. We have set the replication parameter of *Hadoop* cluster to 2.

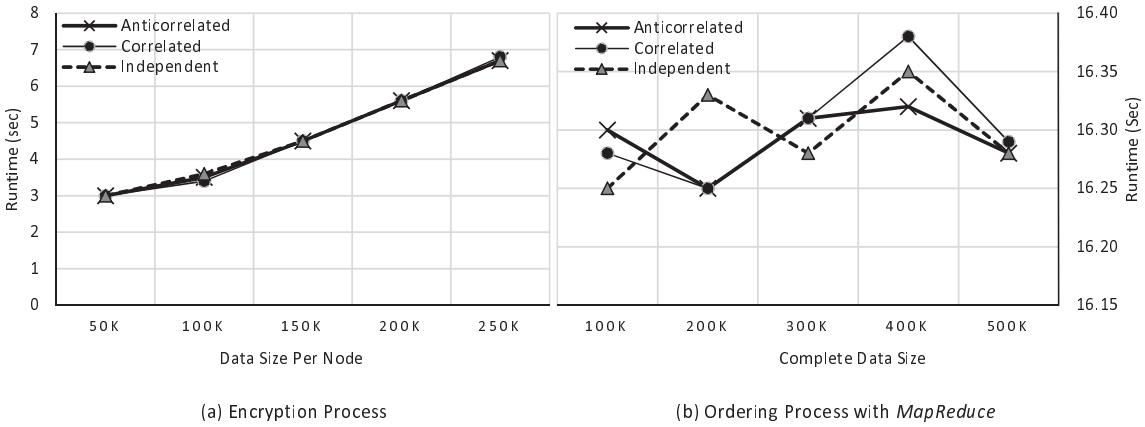


Figure 3.8: Effect of Encryption & Ordering Process

Since none of the existing algorithm has considered the computation issue of secure skyline, we could not compare to others. Instead, we could check the scalability of ours and report a comparison of the time of skyline computation with the security enhancement and the time of skyline computation without the security enhancement. We have used synthetic dataset. The aim of our experiment design was to check computational overhead secure approach over traditional non-secure approach.

A. Effect of encryption process over data distribution: Figure 3.8 (a) shows the effect of encryption process over data distribution. We have varied data size from 50k to 250k per node. Our experimental results for a single iteration of the process explained in Section 3.2.1 shows that the runtimes are identical for Anti-correlated, correlated and Independent dataset. Hence no effect of data distribution upon this step.

B. Ordering with *MapReduce*: Ordering with *MapReduce* is one of the vital time consuming part of the proposed work, described in Section 3.2.2. Figure 3.8(b) shows us the runtime comparison of different data distribution for *MapReduce* ordering. These runtimes are recorded for a single iteration. We varied data size from 100k to 500k. It shows that the ordering time varies very little (in terms of few milliseconds) for different distribution.

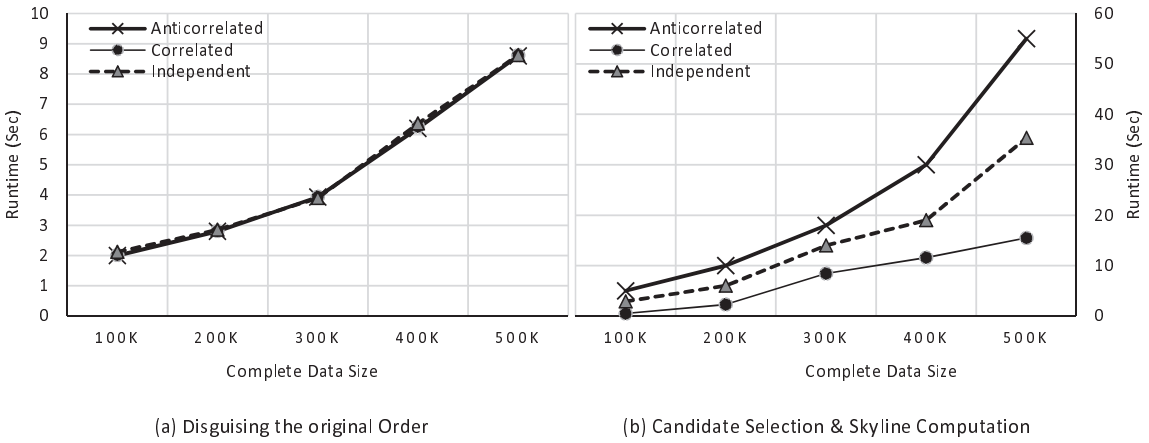
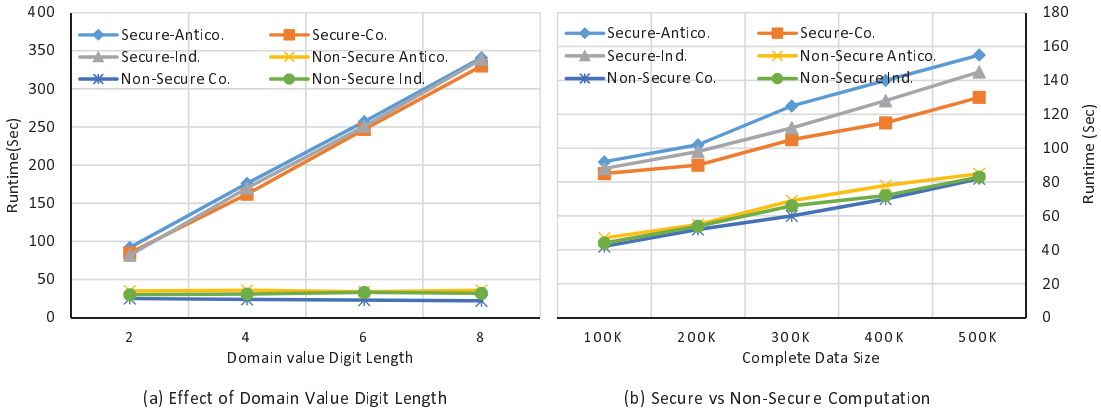


Figure 3.9: Disguising original order, Candidate selection & Skyline computation

C. Disguising the Original Order: Figure 3.9(a) shows the execution time comparison for Disguising the original order among different data distribution. The process is described in Section 3.2.3. Our experimental results shows that this process is not affected by data distribution at all. As shown in figure, we have varied our dataset size from 100k to 500k and found the execution times are identical for Anti-Correlated, Correlated and Independent data distribution.

D. Candidate Selection and Skyline Computation: This process is described in Section 3.2.6. As shown in Figure 3.9(b), this process is affected by data distribution. We varied our dataset size form 100k to 500k and found that this process is more efficient for Correlated dataset and less efficient for Anti-Correlated dataset. However, the performance for Independent dataset lies in between the performance for Anti-Correlated and Correlated dataset.

E. Effect of domain value length: One of the interesting finding for our proposed algorithm is given in Figure 3.10(a). In this experiment we have fixed the data size to 200k and used two dimensional data but varied the domain value digit length and record the execution time for complete calculation. We can see that the domain value digit length has considerable effect on our proposed method. However, in conventional non-secure skyline



(a) Effect of Domain Value Digit Length (b) Secure vs Non-Secure Computation

Figure 3.10: Effect of domain value digit length & Secure *vs* Non-Secure Computation

algorithms like SFS are not affected by such digit length.

F. Secure vs Non-Secure Computation: Figure 3.10(b) gives us an idea about computational overhead due to impose of secure computation scheme. It is obvious that the secure approach requires some extra execution time. However, we can notice that the computational overhead seems to be an identical value though the data size increased linearly. Hence, it may not be considered as overhead when processing "big data". In this experiment we have fixed the domain value digit length to 3 and data dimension to 4.

3.4 Concluding Remarks

This chapter addresses the problem of privacy in distributed skyline query computation. In privacy aware situation, we have to take into account the problem. The author proposed a secure skyline query computation in *MapReduce* framework, which is a popular "big data" computing framework. Through intensive experiments, the author demonstrated the effectiveness and scalability of the proposed algorithm. In future, the author wants to design optimized mechanisms for the proposed secure skyline computation. In addition, the author wants to consider secure computation of other variants of skyline queries, such as k-dominant skyline, k-skyband.

Chapter 4

Secure *k-object* Selection

Top-k queries have been extensively used to make a choice of preferable objects from large dataset. A scoring function and a number k, for number of objects to be selected, are specified by users. Then, top-k query returns k objects based on the user defined scoring function. It is quite possible that scoring functions of every user may not be similar for selecting top-k objects, which indicates that the top-k query results are valuable for those users who share an identical scoring function. And it has already been mentioned that to specify a scoring function user must have some prior domain knowledge. k-object selection query is inspired by the drawbacks of top-k query selection.

In this chapter, the author proposed a k-object selection mechanism that chooses various k objects which are preferable for all users who may have non-identical scoring function; meanwhile, it also ensures the privacy of attribute value during the process of computation.

4.1 k-objects Selection Problem

As illustrated in Figure 4.1, assume that, a symbolic dataset DB is partitioned between two parties: $Party_1$ & $Party_2$ and the partitioned data are private. In order to select the desired result of our k-object query, The author introduces an entity: "*Partially Trusted*

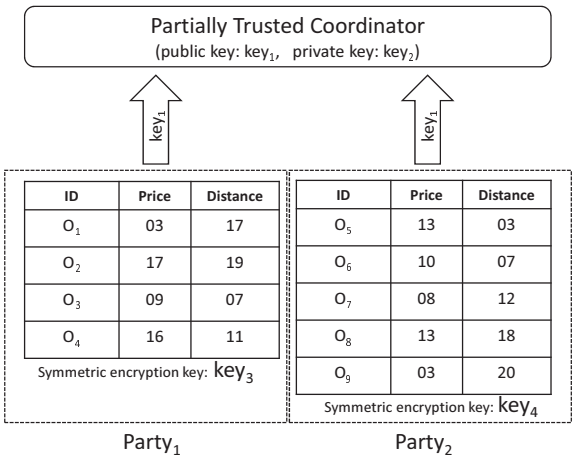


Figure 4.1: Running Example

Coordinator". The "Coordinator" is called "*Partially Trusted*", because it does not disclose any value to other parties. The "Coordinator" is configured to perform *Asymmetric Cryptography*, hence equipped with two encryption keys: one is known as *public key* − key_1, whenever parties send any data to the "Coordinator" it must have been encrypted with key_1. Another is known as *private key* − key_2, data encrypted by key_1 can only be decrypted by key_2. No one but the "Coordinator" knows key_2. Besides "Coordinator"s cryptographic configuration, parties are configured to perform *Symmetric cryptography* too. Hence, each party has its own encryption key, for example, key_3 & key_4 are the *Symmetric encryption* keys for $Party_1$ & $Party_2$ respectively. key_3 & key_4 are known to respected parties only. Our algorithm is based on *MapReduce* framework and our *Partially Trusted Coordinator* plays the role of *master node* of that cluster. In this research work, the author have proposed an efficient secure algorithm, which selects various k objects using *MapReduce*.

4.2 Secure k-objects Selection using *MapReduce*

Proposed algorithm is composed of following five steps: (1) Preparing the $\langle key, value \rangle$ pair, (2) Sorting & disguising order using *MapReduce*, (3) Returning of ordered values, (4)

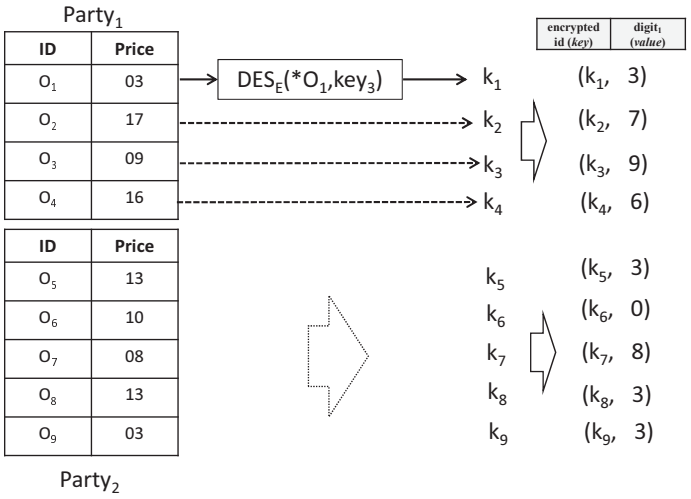


Figure 4.2: Preparing the $\langle key, value \rangle$ pair

Merging, sorting and disguising original rank and, (5) Retrieving the k-object.

4.2.1 Preparing the $\langle key, value \rangle$ pair

In the beginning of explanation, let us note that *key* of $\langle key, value \rangle$ pair has no relation with the term *cryptographic encryption key*. To avoid confusions, from now on, we will denote the *key for cryptographic encryption* only by key_{encr}. Now, as shown in Figure 4.2, each *party* has a collection of two dimensional data which are non-disclosable. Both *parties* are capable to perform *Symmetric cryptography* (e.g. DES -Data Encryption Standard), hence they are equipped with an *encryption key*, which is known to the corresponding *party* only. Each *party* encrypts the ID of data using that *encryption key*. Before encrypting, a predefined text segment (or character stream) is concatenated, as prefix, with the ID of each data object. Reason of such concatenation is simple. This will help the *party* to identify its own data later when the encrypted part will be decrypted again. Therefore, the necessity of searching each *parties'* entire data-space, for identifying its own data ID, will be eliminated. Hence, the performance will be boosted. After encrypting the ID with some prefix, we will get a new ID, let us call it *encrypted_id*, which will be served as *key* (not *encryption key*, but the *MapReduce key-value* pair's *key*) for our consequence processing.

For choosing the *value* parameter, first time, we will use the right most digit of the first dimension. The same process will be repeated for all remaining digits and dimensions too.

To understand the idea of this stage more clearly, let us consider the example illustrated by Figure 4.2. In the given example, we have considered the attribute *price* only. The ID of first data of $Party_1$ is O_1. A prefix "*" is added with O_1 and using its private *encryption key*: key_3 the ID with prefix is being encrypted. Assume that the *encrypted_id* is k_1. Any outsider cannot recognize the original ID from the value k_1 unless he/she has access to *encryption key*: key_3. We have already mentioned that key_3 is known to $Party_1$ only. This *encrypted_id* will serve as *key* in the next stage of processing which will be done in *MapReduce* framework. Now, we will choose the right most digit ($digit_1$) of the attribute *price* as *value*. In the example, *price* value of O_1 is 03, that implies the $digit_1$ will be 3 for O_1. Hence, the $\langle key, value \rangle$ pair for this instance is: $\langle k_1, 3 \rangle$. And for next data object O_2 the $\langle key, value \rangle$ pair will be: $\langle k_2, 7 \rangle$ and so on.

After constructing the $\langle key, value \rangle$ pair, data are sent to the "Coordinator". As we have already mentioned that the "Coordinator" plays the role of *master node* of our *Hadoop* cluster, it processes the submitted data in $MapReduce$ framework. The communication in between the *parties* and "Coordinator" must have been encrypted by key_1, so that only the "Coordinator" can read the submitted $\langle key, value \rangle$ pair by decrypting key_2. The same process is followed for all data in $Party_1$ & $Party_2$.

4.2.2 Sorting & disguising order using *MapReduce*

In this stage, we sort the *encrypted_id*s based on their $digit_1$ values. The actual order of the *encrypted_id* are disguised due to privacy issue − as in some situations order itself a sensitive information. We have done the job in with *MapReduce* framework. It is to be mentioned that we have used the most popular open-source implementation of *MapReduce* − the *Hadoop* framework. The encrypted values of data IDs and corresponding $digit_1$ values are

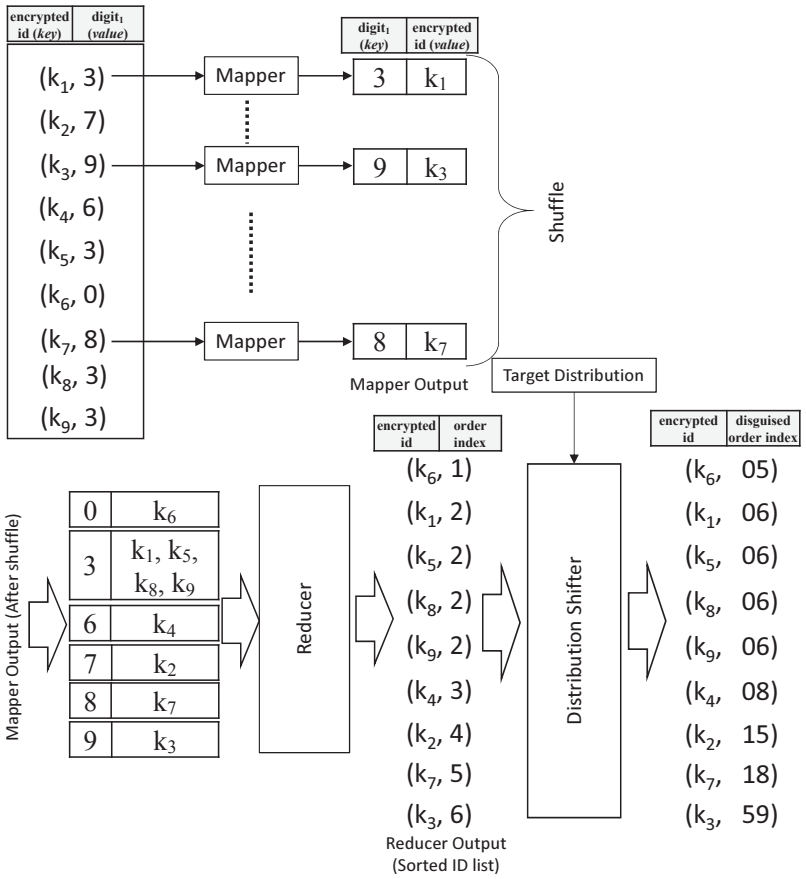


Figure 4.3: Sorting and disguise order using *MapReduce*

stored in distributed file system (*DFS*) more specifically in *HDFS* (*Hadoop Distributed File System*). The *Mapper* read each $\langle encrypted\_id, digit_1 \rangle$ and depict $\langle digit_1, encrypted\_id \rangle$ as *Mapper*'s output. According to the working principle of *MapReduce* framework: the $digit_1$ serves as *key* and the framework itself will shuffle and sort the data; so that the values with similar *key* are tagged together and ordered. The *Reducer* layer collects the shuffled values and produce the sorted order of *encrypted_id*s. It is to be noted that several *encrypted_id*s may have same $digit_1$ value as *key* and they will have the same ranking index.

In Figure 4.3 we can see that the first *Mapper* is taking the $\langle k_1, 3 \rangle$ pair as input and producing $\langle 3, k_1 \rangle$ as output. Similarly the next *Mapper* taking $\langle k_3, 9 \rangle$ and producing $\langle 9, k_3 \rangle$, and so on. After shuffling, values with similar *key* are grouped together and fed into *Reducer* layer and the *Reducer* produces the sorted order of *encrypted_id*. It is to be mentioned

that, this ordered sequence ({1, 2, 3, ...}) is uniformly distributed.

After getting the order index, we need to disguise the actual order. We have already mentioned that the sorted output of order sequence for *encrypted_id* is actually uniformly distributed. Because the default distribution of a order sequence is uniform. In order to hide the actual order ({1, 2, 3, ...}) from intruders, it is a better idea to shift the density of distribution to a target distribution. The details idea of this work is discussed in [2]. In brief it could be expressed as follows: First of all we have to choose a target distribution other than uniform distribution. After choosing the target distribution, we have to generate $|X|$ unique values from the target distribution where X is the collection of ordered sequence indices ($e.g. X = \{1, 2, 3, ...\}$, and for $digit_1$ the maximum value of $|X|$ can be 10, but for next iteration it will be larger: 10^n for $digit_n$). Sort the generated random values into a table T . The sorted i^{th} index value of ordered list (*encrypted_key*) is then given the order value of *T[i]*.

If we look at the Figure 4.3, our *Distribution Shifter* module receives the target distribution and generate 6 unique (because, $|X| = 6$) values and after ordering T= {*05, 06, 08, 15, 18, 59*}. From previous stage, we already know that k_6 is ranked 1 in our sorted list. However, we will replace this index with the first value of T. *i.e.* the new order index of k_6 is 05. Similarly k_1, k_5, k_8 & k_9's order indices will be 06 and so on.

Hence we have been able to shift the order index while preserving the order sequence.

4.2.3 Returning of ordered values

Transferring to target distribution ensures us that the order indices of *encrypted_id*s are not disclosing the actual order information while we are also preserving the original order sequence. After re-indexing the order-indices, results are sent back to both *parties*. As the actual IDs are encrypted and order-indices are disguised, it's not possible for $Party_1$ or any third party intruder to retrieve any information of $Party_2$ or vice-versa. After

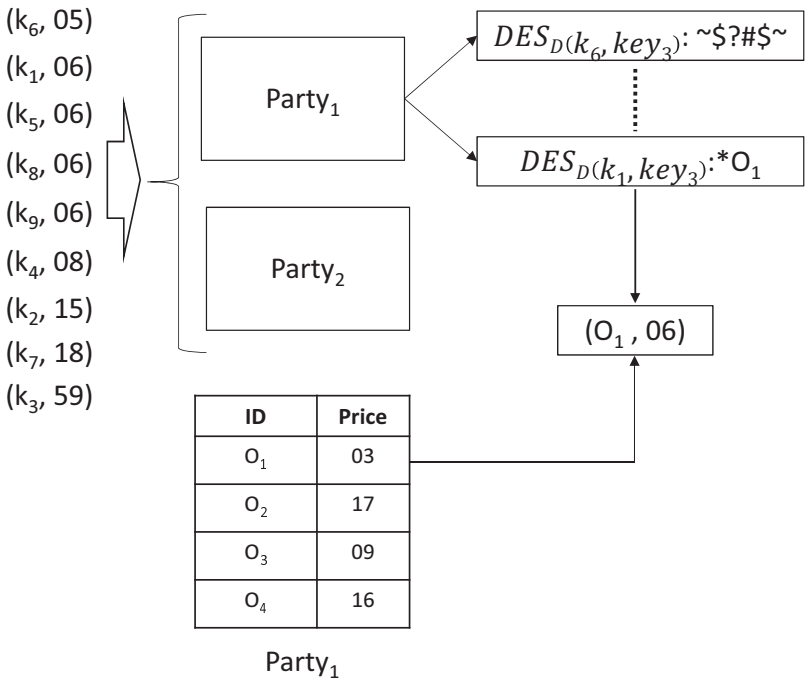


Figure 4.4: Decryption and gathering order

receiving the result from previous stage, each *parity* tries to decrypt the *encrypted_id* with its own *encryption key*. If the data belongs to the corresponding *party*, after decryption, the *party* can easily identify it by checking the prefix it added in time of encryption. If the decrypted ID does not start with the original prefix, it does not belong to that *party*. So the *party* dispose that ID. If the prefix matches, the ID is searched and corresponding order index is stored into its data repository. It is possible that, due to some coincidence, several *parties* may have used same prefix or after decryption the decrypted ID is matched with *party*'s original prefix. In that case the ID search throughout the data repository will go in vain. It is difficult to find some way to avoid such coincidence. In other situations, prefix addition will enable us to avoid unnecessary searching for deciding whether or not this ID belongs to current *party*.

In Figure 4.4, when $Party_1$ tries to decrepit k_6 with its own encryption key: key_3, it gets some garbage values (not starting with "*"). That indicates: k_6 does not belong to $Party_1$. However, when k_1 is decrypted, it results $*O_1$, where "*" is the original prefix

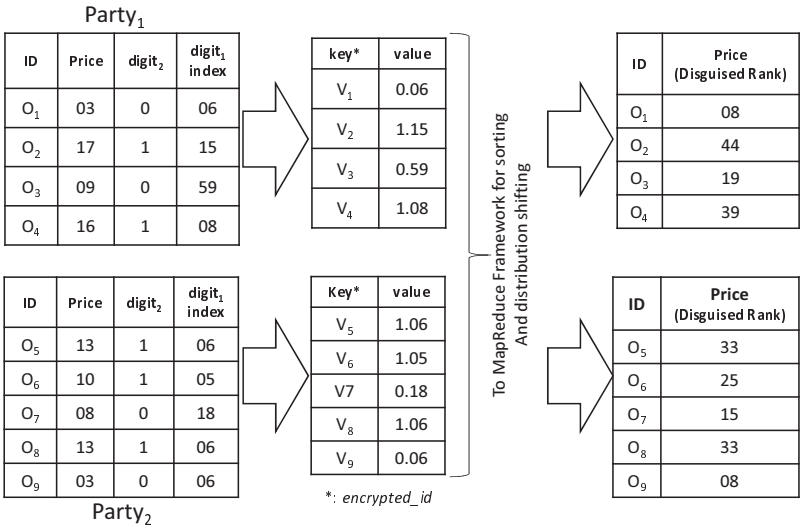


Figure 4.5: Merging, sorting and disguising original rank

added by $Party_1$. Now the entire data repository is searched for O_1 and when found then O_1 's disguised order value 06 for *price* attribute's $digit_1$ is preserved for next stage use.

4.2.4 Merging, sorting and disguising original rank

After receiving the order indices for $digit_1$, each *party*'s next job is to merge those values with $digit_2$ values. These new values along with encrypted *ID*s are sent to *MapReduce* framework for sorting to get the actual order. The encryption of *ID*s can be done by using previous *encryption key*. But it is better to perform encryption with a new key_{encr}. The idea of adding a prefix (described in section 4.2.1) is also applicable here. If we add new character stream as prefix, using the old *encryption key* may also be acceptable. Again, as the actual order will be uniformly distributed, the process described in section 4.2.2 will be used to disguise the order rank.

In the Figure 4.5, we can see in $Party_1$, O_1 has been encrypted to a new value V_1. We can also see that the $digit_1$ index of *ID* O_1 is 06 and $digit_2$ value is 0. And from that information we construct the next value of *price* attribute as 0.06. Similarly, for *ID* O_2 the next value is 1.15. After calculating all the next values of *price* attribute, we feed these

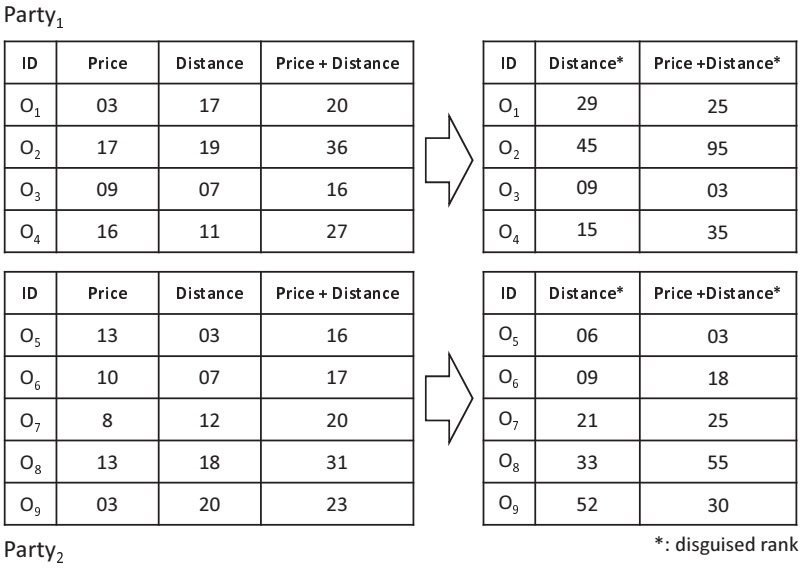


Figure 4.6: Disguised rank for *distance* & (*price* + *distance*)

data to *MapReduce* framework to get disguised order of ID's. The process is similar to what we have explained in section 4.2.2. We have to calculate the disguised rank of other attribute (e.g. distance) values too. Figure 4.6 shows the calculative result of the process for *distance* and (*price* + *distance*) attribute. These orders will be used to calculate the candidate of our result.

4.2.5 Retrieving the k-objects

Now, we have to find a list of candidate for *skyline* from the disguised attribute ranking indices. For better understanding, in this section we are presenting the data *ID*s in plain text format. In actual situation they will be encrypted by the corresponding parties. The process of getting candidates *skyline* has originally proposed in one of our previous work, but we omitted the reference for blind review. In the work, we have partitioned the dataset vertically and sort each partition. That means, we have to sort the data objects according to attribute values. Based on the result of sorting, the object *ID*s are given a ranking value. A *candidate finding module* collects top *ID*s from each attribute horizontally. The *module* maintains counter for each retrieved object. When a counter value becomes equal to the

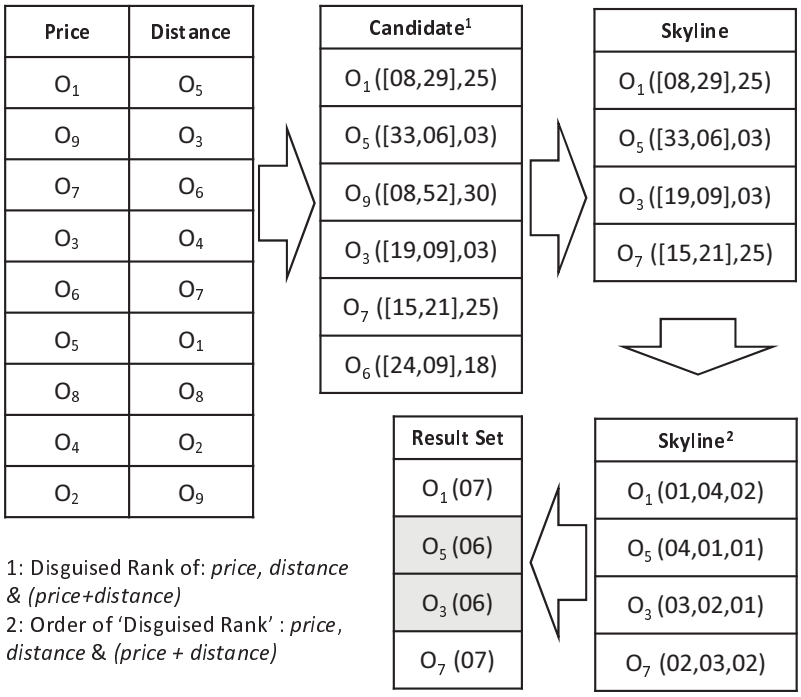


Figure 4.7: Retrieving k-objects

number of attributes, then it stops *ID* collection. The IDs collected by the *module* are candidate of *Skyline* query result. In the Figure 4.7, data *ID*s are already sorted according to *price* and *distance*. In first iteration, a *candidate finding module* will pick $\{O_1, O_5\}$ – as they are the top ranked objects. In second iteration $\{O_9, O_3\}$ will be picked. In the fourth iteration, the collecting procedure will be stopped as the counter value for O_3 equals 2 which is the same as the number of attribute.

As the *candidate finding module* stops, we get the list of candidate skyline as: $\{O_1, O_5, O_9, O_3, O_7, O_6\}$. The list of *candidate skyline* comes with the *disguise ranking* indices of original attributes: *price* & *distance* and calculative attribute: $(price + distance)$. From these *disguise ranking* indices we have to find the *skyline*. As the number for *candidates* are not so high as the original data, any *skyline* computation algorithm can perform the computation easily. The *skyline* query will be calculated based on the *disguise ranking* indices of original attributes: *price* & *distance* only. In the Figure 4.7, we can see that $\{O_1, O_5, O_3, O_7\}$ be the result of

skyline query.

After finding the result of *skyline* query, it is easy to find the 'order of *disguise rank*' i.e. the ranking of *disguise ranking*. The aggregation of these 'order of *disguise rank*' are considered as ranking value for our k-object selection problem. In our Figure 4.7, a candidate object O_1 has *disguise rank* $08, 29$ & 25 for *price, distance* & $(price + distance)$ respectively. Similarly, O_5 has $33, 06$ & 03 and so on. Using these *disguised ranking* indices, we can find the order of *disguised ranking*: for $O_1 - \{01, 04, 02\}$ [because 08 is the smallest *disguised rank* index in *price* attribute, 29 is the fourth in *distance* and 25 is the second in $(price + distance)$ attribute], similarly for $O_5 - \{04, 01, 01\}$ and so on.

When we calculate the sum of these indices, we will get our scoring values. As shown in the Figure 4.7, O_1's scoring value 07 [e.g. $07 = 01+04+02$], O_5's 06 [e.g. $06 = 04+01+01$] and so on. In our previous work [37], we have shown how to calculate the scoring value using *MapReduce* framework.

If user sets the query for $k = 2$ then object O_3 & O_5 will be retrieved [since O_3 & O_5's *sum (order of disguised rank)* be the smallest two] as a result of our *k-object* selection problem.

4.3 Performance Evaluation

This section reports the experimental results to validate the effectiveness and efficiency of proposed method. We set up a cluster of 4 commodity PCs in a high speed gigabit networks, each of which has an Intel Core 2 Duo E8500 3.16 GHz CPU, 8 GB memory. These machines are connected with a Gbps LAN connection. 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.

We have conducted a series of experiments with different data distributions, dimensionalities, data cardinalities and changing attribute values' digit length to evaluate the

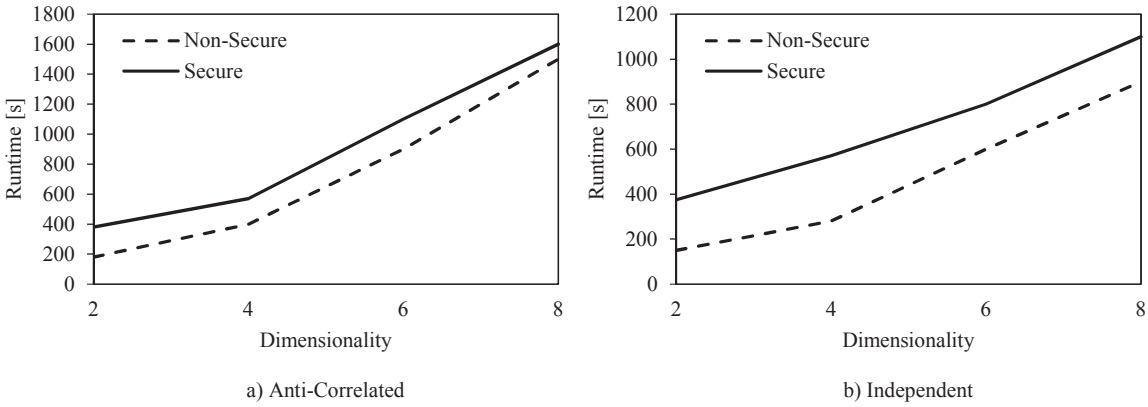


Figure 4.8: performance for different data dimension

effectiveness and efficiency of our proposed method. When designing experiments, our main concern was to find the overhead of runtime costs needed for imposing secure computation with respect to non-secure approach considered in [37]. It is certain that our proposed secured algorithm will take some extra time than the conventional non-secure approach defined in [37].

The data cardinality mentioned in our experiment is the size of total data distributed among several parties. Each experiment is repeated five times and the average result is considered for performance evaluation. Two data distributions are considered as follows:

Anti-Correlated: an anti-correlated dataset represents an environment in which, if an object has a small coordinate on some dimension, it tends to have a large coordinate on at least another dimension.

Independent: for this type of dataset, all attribute values are generated independently using uniform distribution. Under this distribution, the total number of non-dominating objects is between that of the correlated and the anti-correlated datasets.

4.3.1 Effect of Dimensionality

In Figure 4.8, we record the effect of dimensionality on runtime cost. We fix the data cardinality to 100k and vary dataset dimensionality ranges from 2 to 8. The runtime costs

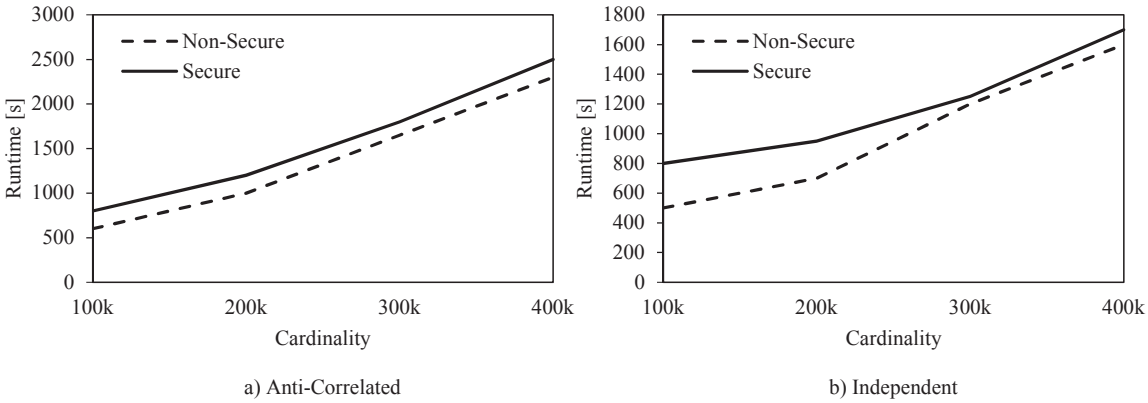


Figure 4.9: Performance for different cardinality

of the algorithm increase along with the increase of dimensionality, which is quiet usual. Runtime costs for secure computation is higher than the non-secure computation process that we have proposed in our previous work [37]. For Anti-Correlated data, the increase of cost is about hundred percent in lower and twenty percent in higher dimension. However, increase of cost for Independent data is about fifty percent in higher dimension. Though the percentage of runtime execution is significantly high, it is to be noted that, for both Anti-Correlated and Independent data, the difference between secure and non-secure execution times are almost a constant value. Which indicates that secure computation is more useful for higher dimension computation, which is quite impressive. The run-time results for this experiment are shown in Figure 4.8 (a) and (b) for Anti-correlated and Independent data respectively.

4.3.2 Effect of Cardinality

In Figure 4.9, we record the effect of cardinality on runtime cost. For this experiment, we fix the data dimensionality to 4 and vary dataset cardinality ranges from 100k to 400k. The runtime costs of the algorithm increase along with the increase of cardinality, which is as expected. For both Anti-Correlated and Independent dataset, the runtime cost overheads are not significantly higher than the conventional approach. Moreover, in case of Anti-

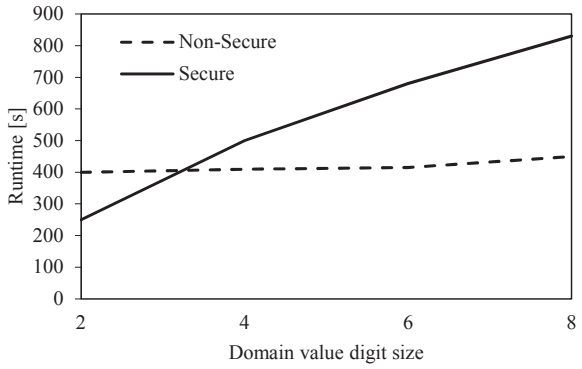


Figure 4.10: Effect of changing maximum length of attribute value digit

Correlated data, it seems to be almost identical. The experiments suggest that for higher cardinality, our secure approach will not cost much overhead than previously proposed non-secure approach [37].

Figure 4.9 (a) and (b) shows the performance on anti-correlated and independent datasets respectively.

4.3.3 Effect of Digit length of values

In Figure 4.10, we record the effect of domain value digit size on runtime cost. For this experiment, we fix the data dimensionality to 4 and cardinality to 100k. We vary the domain value digit size of dimensions from 2 to 8. We can see that our proposed algorithm is heavily affected by the length of attribute value digits. It is because of the multiple execution of steps described in 4.2.2 and 4.2.3. The execution of steps 4.2.2 and 4.2.3 are directly proportional to the length of attribute digit. From experimental result we do confirm that the traditional approach is free from digit length effect, however the performance of our proposed approach is highly affected by the length of attribute value. Hence, in case of higher digit length values, its not suitable if performance is preferred over security.

4.4 Concluding Remarks

This research work addresses k-object selection problem and present a secure distributed algorithm for selecting k objects that are preferable for all users who may have different preference. During the computation we have ensured the privacy of data. Hence multiple parties can participate without disclosing their sensitive data. The idea of skyline query along with perturbed cipher have been used to select k objects and proposed an efficient secure *MapReduce* algorithm. Effectiveness of the proposed algorithm can be verified by experimental results. It is possible to extend this work in a number of directions. First, from the perspective of parallel computing, how to compute k-object from streaming dataset. Secondly, to design an efficient index based (R-tree/B-tree) algorithm are promising research topics.

Chapter 5

Mining Social Media

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

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 interpersonal 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 5.1. 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

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 5.1: Example of Facebook data

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 5.1, 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 5.1 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 "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 chapter, 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 [8, 19, 42]. For better understanding and simplicity, we tried to keep the *MapReduce* explanation figures as simple as possible. In summary, the contributions of this work 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 chapter is organized as follows. Section 5.1 presents the notions and properties of key person computation using skyline query. We explained the detailed algorithm with appropriate examples and analysis in Section 5.2. We experimentally evaluate the algorithm in Section 5.3 under a variety of settings. Section 5.4 concludes the chapter.

5.1 Preliminaries

In this section, we present definitions and basic properties of our key person selection problem. Let us assume that Table 5.1 shows the calculated values of the data described in Figure 5.1.

5.1.1 Social Network Metrics

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

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 5.1: Calculated Example Data

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:

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

More friend counts indicate higher friend power. For example, in Table 5.1 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:

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

More followers mean better followee strength. Table 5.1 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 5.1, 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 5.1, 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 5.1, 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 5.1.

5.1.2 Dominance and Skyline

Let us assume we have a dataset DS (shown in Table 5.1) 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 5.1 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 5.1, users "E" and "F" are not dominated by any other user. Therefore, they represent the key person skyline result of the dataset DS.

5.1.3 Comment Bias

"Opinion mining," also known as "sentiment analysis," [27] 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 [31]:

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.

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

5.2.1 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

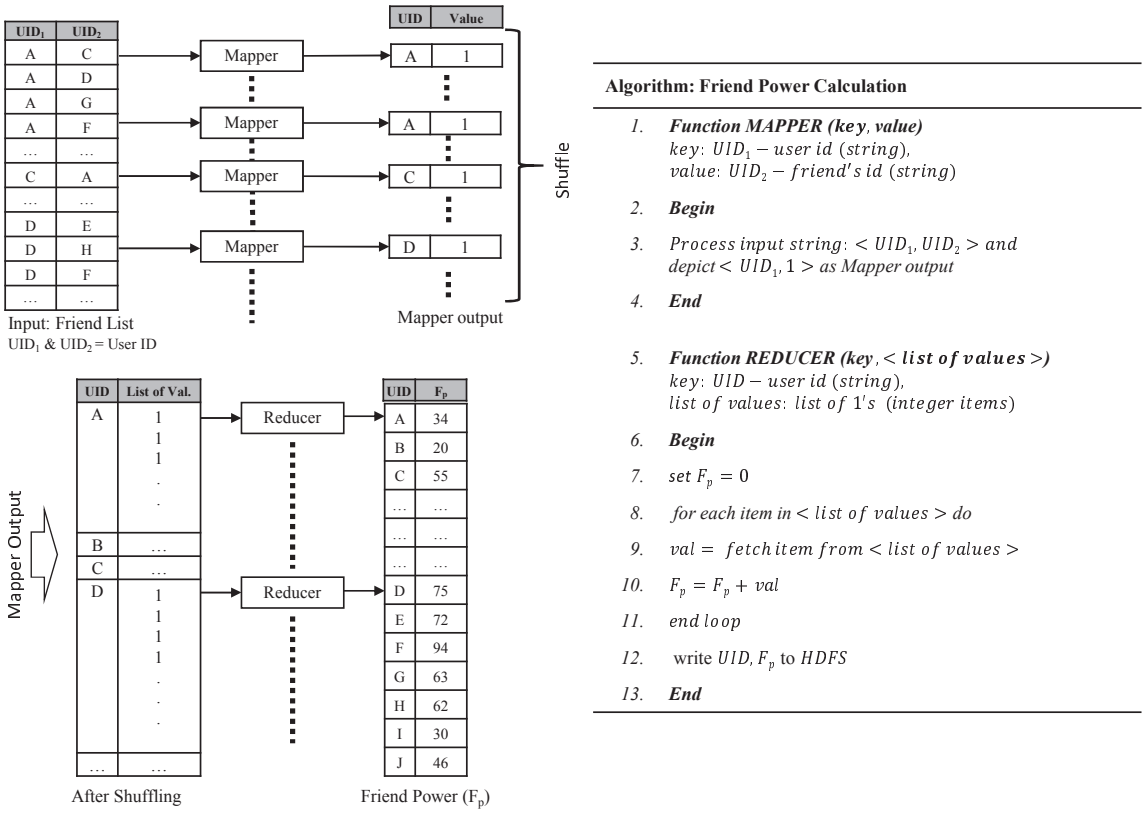


Figure 5.2: Friend power computation procedure

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 5.2 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.

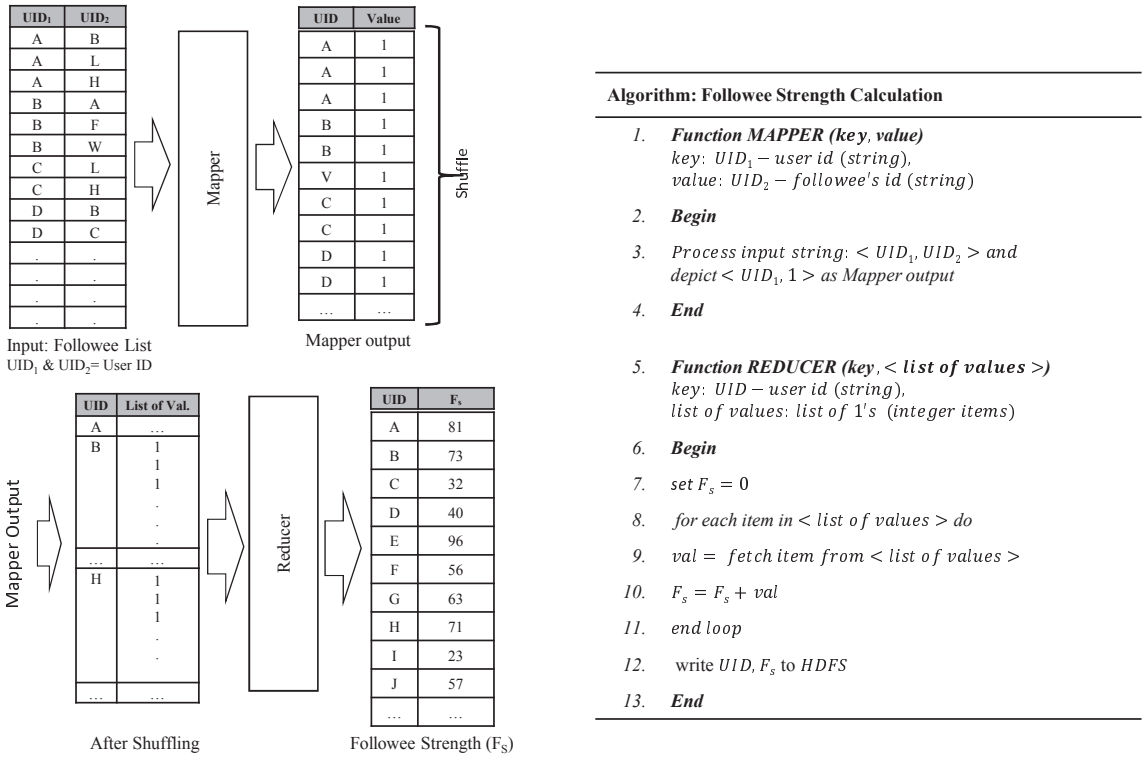
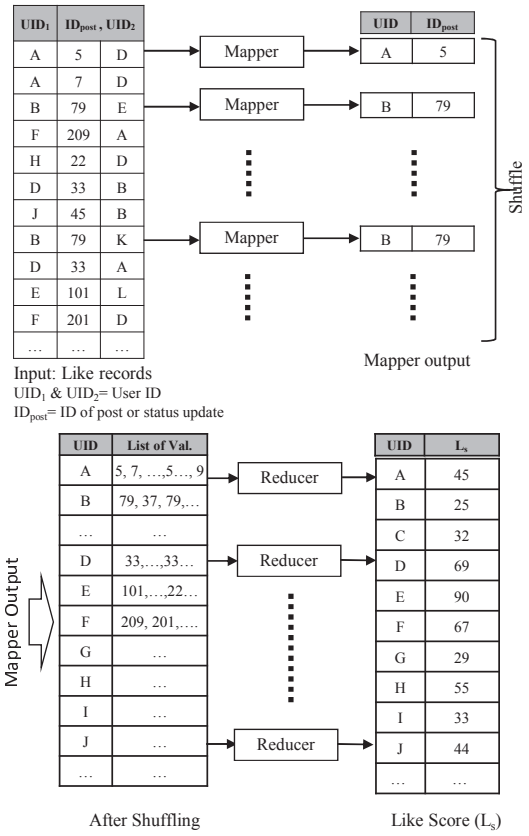


Figure 5.3: Followee strength computation procedure

5.2.2 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 $DataNodes$. 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 5.3 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.



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 5.4: Like Score (L_s) calculation procedure

5.2.3 Like Score (L_s)

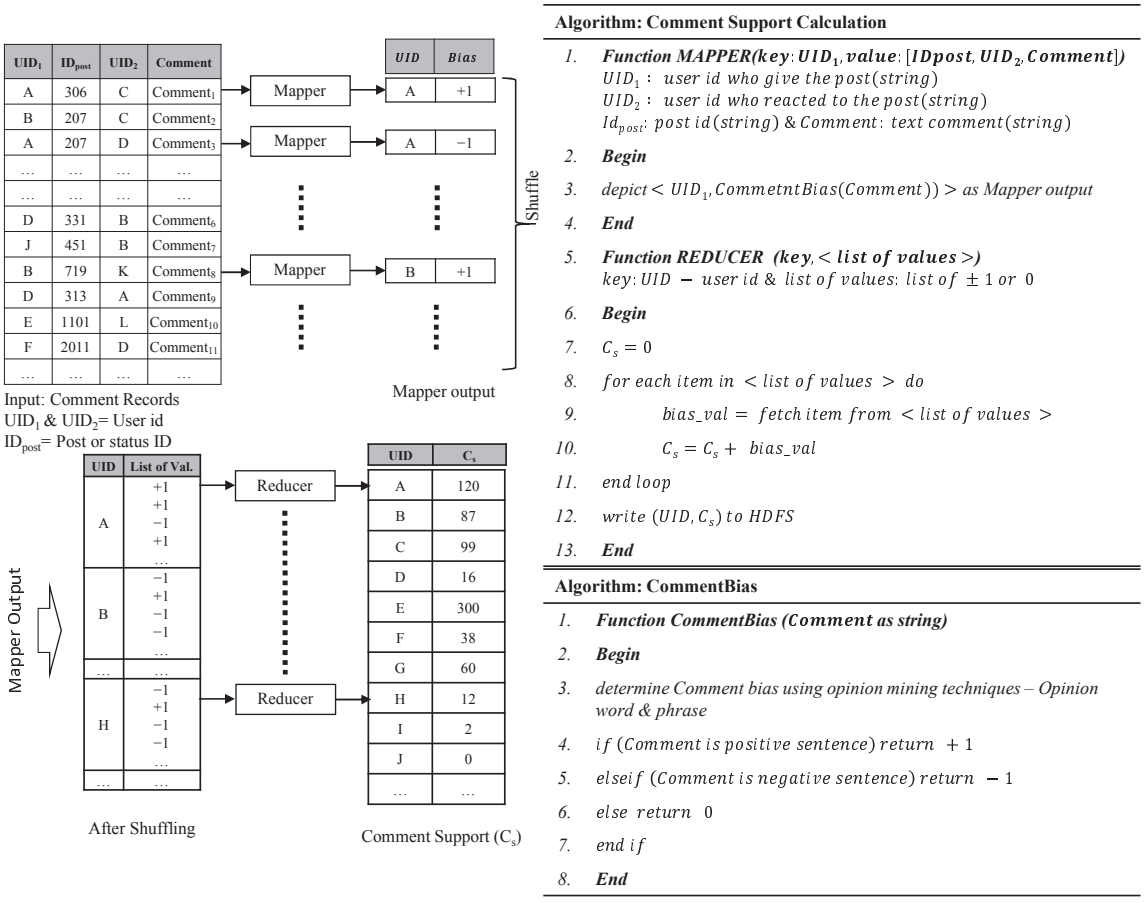
"Like" score (L_s) is one of the important matrices of a social network like Facebook. This section explains the procedure of calculating L_s in *MapReduce* framework. Here, we assume that the kvs format of our input data is structured as: $\langle UID_1, (ID_{post}, UID_2)\rangle$,

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 5.4 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 with similar UID. In Figure 5.4, 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.

5.2.4 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 sta-



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* $\pm 1\ or\ 0$
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***

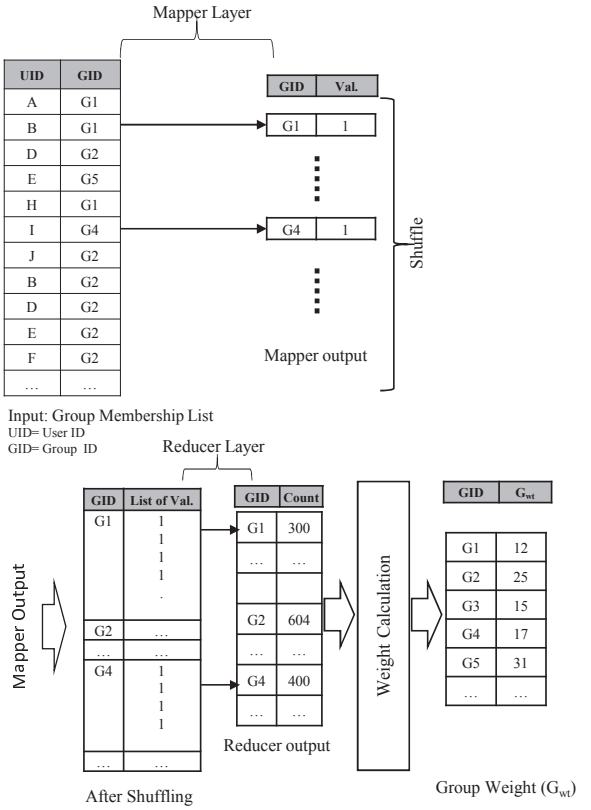
Figure 5.5: Comment support (C_s) calculation procedure

tus 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 5.5 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 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 5.5.



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 5.6: Group weight computation procedure

5.2.5 Group Weight (G_{wt})

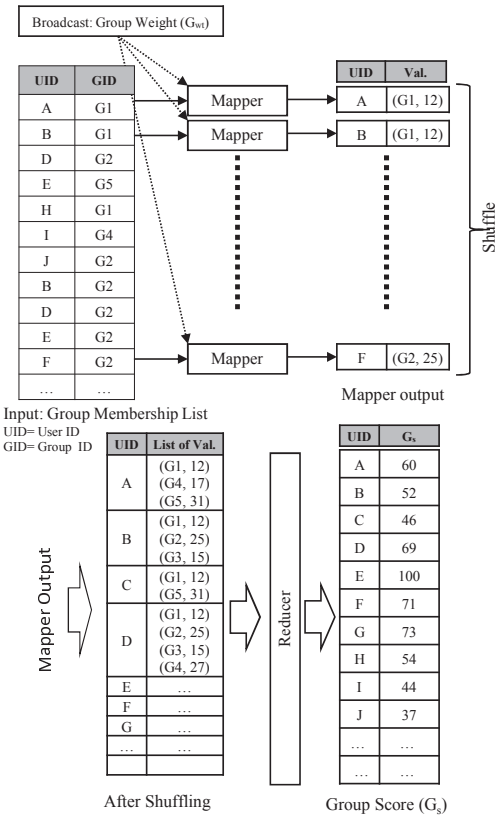
To calculate group weight, each mapper takes UID and their corresponding group lists as input. Each Mapper generates a pair of intermediate output: $\langle GID, value\rangle$. Here, GID represents group id. After shuffling and grouping by the corresponding GID, the *Mapper*s outputs are sent to *Reducer*s. 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 5.6 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.

5.2.6 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 5.7. 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 5.7



Algorithm: Group Score Calculation

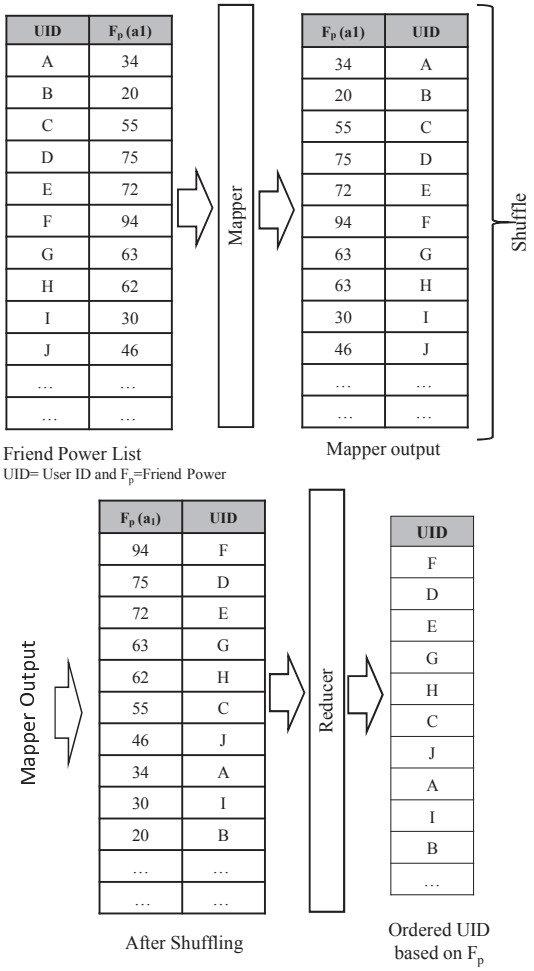
1. Broadcast G_{wt} to each *Mapper*
2. ***Function MAPPER*** $(\boldsymbol{key}, \boldsymbol{value})$
   $key: UID - user\ id\ (string)$
   $value:\ GID - group\ id\ (string)$
3. ***Begin***
4. $process < UID, GID >$ *and*
   $depict < UID, (GID, G_{wt}) >$ *as Mapper output*
5. ***End***
6. ***Function*** **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 5.7: Group score computation procedure

5.2.7 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, 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 5.8 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 5.8, 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



Algorithm : Ordering with MapReduce

1. ***Function MAPPER*** $(key, value)$
   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** $(key, < list\ of\ values >)$
   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 5.8: Sorting on friend power

class of *Hadoop* (because the default sorting operation of *Hadoop* framework is ascending). For clear understanding, we have included a formal algorithm in Figure 5.8.

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 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 5.9 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 [38]. For interested audiences, we briefly describe the idea of that work below.

In the work [38], 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 5.9, *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.

In the example, the domain value of "F" for five domain F_p,F_s, L_s, C_s & G_s are

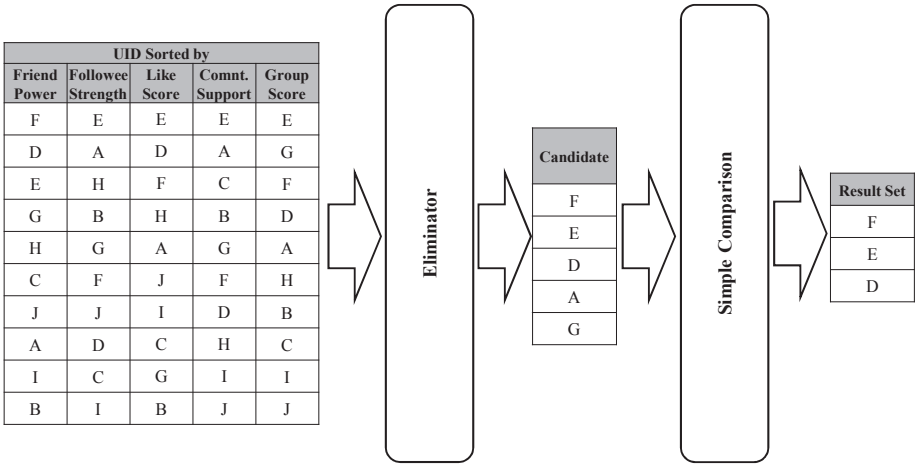


Figure 5.9: Skyline computation

$\{94, 56, 67, 38$ & $71\}$, respectively. Table 5.2 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 5.2: 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*.

5.3 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

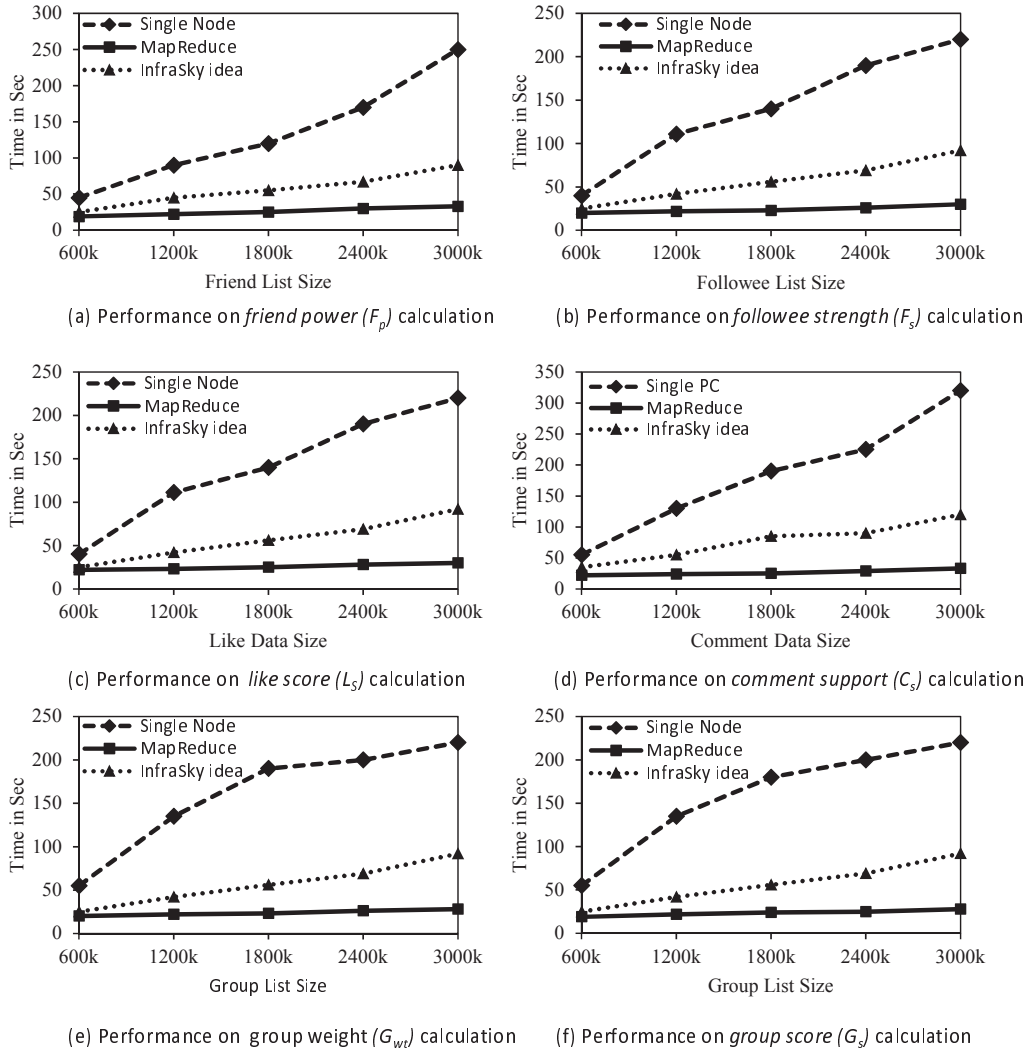
memory. These machines were connected with *Cisco SG300-20* gigabit manageable switch. We compile the source codes under Java V8. We used *hadoop* version 2.5.2 and the OS platform was 64 bit CentOS 7. The replication parameter of *hadoop* configuration was 2.

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 [34], 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.

5.3.1 Effect of Proposed Algorithm in *MapReduce*

We study the effect of various steps described in Section 5.2. Figure 5.10 (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 [34], 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 5.10, 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 [34]. 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.



(a) Performance on *friend power* (F_p) calculation

(b) Performance on *followee strength* (F_s) calculation

(c) Performance on *like score* (L_S) calculation

(d) Performance on *comment support* (C_s) calculation

(e) Performance on group weight (G_{wt}) calculation

(f) Performance on *group score* (G_s) calculation

Figure 5.10: Performance of domain value calculation

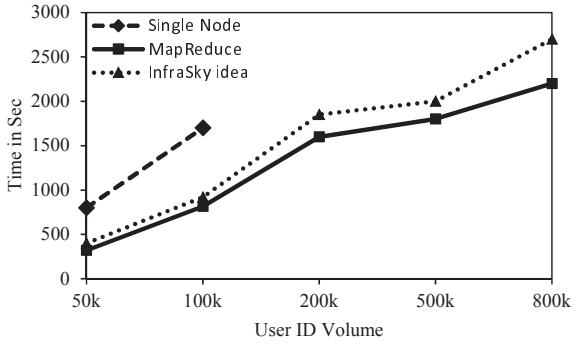


Figure 5.11: Performance on skyline computation

5.3.2 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 5.11. 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 [34] was implemented in our proposed *MapReduce* framework, therefore, the execution time is identical.

5.3.3 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 work can minimize the

possibility of encountering such a problem. In our previous work [47], we have used three social media matrices: friend power (F_p), followee strength (F_s) and group score (G_s). However, in this research 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 5.12. It is clear that if we use the conventional three dimensional approach [47], 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.

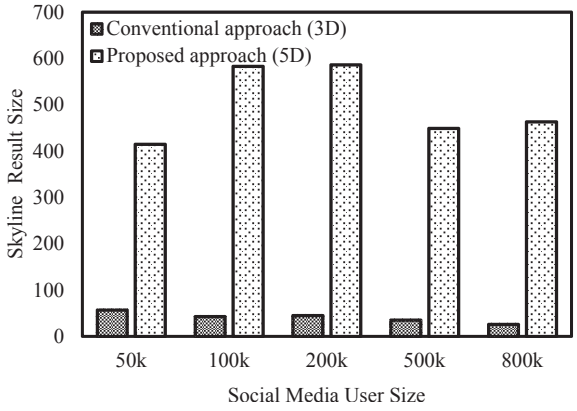


Figure 5.12: Effect of introducing two new Social Media metrics

5.4 Concluding Remarks

In this study, the author addressed the problem of selecting key persons from different groups of Facebook network. The author considered 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 proposed approach the author considered 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.

Chapter 6

Conclusion

In this chapter, the author first discussed the application of proposed models in Section 6.1. Then in Section 6.2 the author discussed key contribution. And finally, in Section 6.3, the author viewed the future scope for improving the current models.

6.1 Applications of proposed models

For last few decades, the *skyline* query is known to be a popular query algorithm for finding interesting data. Proposed model (in Chapter 3) finds the result of *skyline* query without disclosing domain values. Definitely, it will create opportunity for finding interesting objects where data belongs to multiparty and parties are not interested to disclose the domain values at all. The model is designed by using Google's *MapReduce* framework, which gives users to deploy the model in situations where conventional single core algorithms are not suitable.

Model proposed in Chapter 4 is about secure k-object selection problem. Top-k query is also a well known query. Such query requires some ranking or scoring function provided by users. To define ranking/scoring function users have to have some domain knowledge. Conventional k-object selection algorithms allow users to find top k objects without having

any prior domain knowledge. But conventional approach is not useful when data belongs to several parties who are not willing to share though want to provide answer of users query. However, proposed model will be useful in such situations.

Social medias, like Facebook & Twitter, are source of gigantic amount of data. These data can be used to understand mass people more closely. Such enormous amount of data can also be used to manipulate people opinion, their way of thinking. One way of manipulation is to find some influential social media users and use them as brand ambassador of some product or some opinion. Proposed model, in Chapter 5, explains a way to find some key person of social media using *skyline* query on *MapReduce* framework. As explained in Chapter 5, our proposed model uses five social media matrices to find the key persons.

6.2 Contribution

Computational efficiency, as well as privacy of data have been received considerable attention from database research community for decades. In this research work, the author studied three sophisticated aspects of *skyline* queries and its variants: (i) Secure skyline query (ii) Secure k-object selection and (iii) Finding key person of social media. Main contributions are stated as follows:

6.2.1 Contribution on Problem I

Secure *skyline* query problem is described in Chapter 3. *Skyline* query is considered as one of the most popular and useful query for last few decades in database research community. Though the importance of such query is clear to researchers, conventional *skyline* query algorithms are unable to deliver any result when domain values of data objects are hidden or non-disclosable. The problem becomes worst when the authority of data belongs to several parties and they are not willing to share or disclose its domain values but want to find the result of *skyline* query. The author proposed a novel model where parties can

compute the result of *skyline* query without disclosing a single domain value. However, proposed model is designed by using Google's *MapReduce* framework - which enables the model's capability of processing enormous amount of data in distributed environment. Experimental result ensures that it will consume almost a constant overhead while applying secure computation approach instead of non-secure conventional approach. Such constant overhead is acceptable while considering the issue of data privacy.

6.2.2 Contribution on Problem II

k-object selection problem was defined as alternative to top-k query. In top-k query user must define some scoring functions. But in situation where users do not have any domain knowledge, conventional top-k query is useless. k-object selection query can help such users. However, in the situation where data belongs to several parties and parties are not willing to disclose any domain value but want to get result of k-object query: conventional k-object selection algorithm will not work. In Chapter 4, proposed model has addressed the issue. Proposed model is capable to retrieve k-object query result without having exact domain values from different parties. This model is also designed by using Google's *MapReduce* framework to ensure its capability of processing BigData. Experimental results also support capability of the model.

6.2.3 Contribution on Problem III

Social medias are sources of gigantic amount of data. These enormous amount of data can be used to understand mass people in better way. Opinion of mass people can also be manipulated by using such huge amount of data. The basic idea of successful manipulation comes from selecting or finding some key person from social media. The author proposed a model of finding key persons from social media. Though our proposed final model uses five social media matrices the predecessor model used three matrices. In both models we

have designed *skyline* query to identify some key persons from social media. Though the conventional social media mining problems are considered as Graph Mining problem, which are analogous to top-k query, this dissertation focused the issue from a new direction of thinking using *skyline* query's dominance check approach. Our experimental results showed that the performance of final model was better than its predecessor.

6.3 Future Direction

This thesis suggests several promising direction for future research.

6.3.1 Secure *skyline* query

The secure *skyline* query issue addresses the problem of privacy in distributed skyline query computation. In privacy aware situation, we have to take into account the problem. The author proposed a secure skyline query computation in *MapReduce* framework, which is a popular "big data" computing framework. Through intensive experiments, we demonstrated the effectiveness and scalability of the proposed algorithm. Its possible to design optimized mechanisms for the proposed secure skyline computation by minimizing the total number of *MapReduce* job submitted to *Hadoop* framework. Secure computation of other variants of *skyline* queries, like k-dominant skyline or k-skyband can also be considered in future.

6.3.2 Secure k-object selection

Secured k-object selection problem was designed and presented as a secure distributed algorithm for selecting k objects that are preferable for all users who may have different preference. Privacy of data have been ensured during for the computation of multiparty non-disclosable data. Hence multiple parties can participate without disclosing their sensitive data. The idea of skyline query along with perturbed cipher have been used to select

k objects and proposed an efficient secure *MapReduce* algorithm. Experimental results ensured the effectiveness of the proposed algorithm. There exists some scopes to extend this work in a number of directions. First, from the perspective of parallel computing, how to compute k-object from streaming dataset. Secondly, to design an efficient index based (R-tree/B-tree) algorithm are promising research topics.

6.3.3 Finding key person problem

In "Finding key person problem" study, we addressed the problem of selecting key persons from different groups of Facebook network. The author proposed a novel model for identifying key persons. The main feature of our model is: it uses skyline query to retrieve results. Moreover, proposed model consider the *MapReduce* framework to speed up the computation process and to handle massive data in distributed manner. 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 can 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, users of this model may get confused to select appropriate key persons. In such case we need to consider other variant queries such as representative skyline query and top-k query.

Reference

[1] F. N. Afrati, P. Koutris, D. Suciu, and J. D. Ullman. Parallel skyline queries. In *ICDT*, page 274284, 2012.

[2] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *ACM SIGMOD International Conference on Management of Data*, pages 563–574, 2004.

[3] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *ACM SIGMOD International Conference on Management of Data*, pages 439–450. ACM, 2000.

[4] 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.

[5] Apache. Apache hadoop. In *http://hadoop.apache,org*. 2010.

[6] Mohammad Shamsul Arefin and Yasuhiko Morimoto. Privacy aware parallel computation of skyline sets queries from distributed databases. *2013 International Conference on Computing, Networking and Communications (ICNC)*, pages 186–192, 2011.

[7] Wolf-Tilo Balke, Ulrich Güntzer, and Jason Xin Zheng. *Efficient Distributed Skylining for Web Information Systems*, pages 256–273. Springer Berlin Heidelberg, 2004.

[8] 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 *SIGMOD*, pages 975–986, 2010.

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

[10] 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.

[11] Kevin Chen-Chuan Chang and Seung-won Hwang. Minimal probing: Supporting expensive predicates for top-k queries. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, SIGMOD '02, pages 346–357, New York, NY, USA, 2002. ACM.

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

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

[14] Lakhmi C. Jain Editors: Barbara Catania. *Intelligent Systems Reference Library Volume 36 2013*. Springer Berlin Heidelberg, 2013.

[15] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '01, pages 102–113, New York, NY, USA, 2001. ACM.

[16] 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.

[17] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229. ACM, 1987.

[18] Seung-won Hwang and Kevin Chen-Chuan Chang. Optimizing access cost for top-k queries over web sources: A unified cost-based approach. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, pages 188–189, Washington, DC, USA, 2005. IEEE Computer Society.

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

[20] M. Kantarcioglu and O. Kardes. Privacy-preserving data mining in the malicious model. *International Journal of Information and Computer Security (IJICS)*, 2(4):353–375, 2008.

[21] Hua Lu Kasper Mullesgaard, Jens Laurits Pedersen and Yongluan Zhou. Efficient skyline computation in mapreduce. In *EDBT*, pages 37–48, 2014.

[22] Donald Kossmann, Frank Ramsak, and Steffen Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 275–286. VLDB Endowment, 2002.

[23] 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.

[24] Zhensong Liao, Qiang Yin, Yan Huang, and Li Sheng. Management and application of mobile big data. *Int. J. of Embedded Systems (IJES)*, 7(1):63–70, 2015.

[25] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 86–95, April 2007.

[26] Yehuda Lindell and Benny Pinkas. *Privacy Preserving Data Mining*, pages 36–54. Springer, 2000.

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

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

[29] Amélie Marian, Nicolas Bruno, and Luis Gravano. Evaluating top-k queries over web-accessible databases. *ACM Trans. Database Syst.*, 29(2):319–362, June 2004.

[30] Yasuhiko Morimoto, Mohammad Shamsul Arefin, and Mohammad Anisuzzaman Siddique. Agent-based anonymous skyline set computation in cloud databases. *Int. J. of Computational Science and Engineering(IJCSE)*, 7(1):73–81, 2012.

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

[32] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82, March 2005.

[33] Yoonjae Park, Jun-Ki Min, and Kyuseok Shim. Parallel computation of skyline and reverse skyline queries using mapreduce. In *Proceedings of the VLDB Endowment*, volume 6-14, pages 2002–2013, August 2013.

[34] 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.

[35] João B. Rocha, Akrivi Vlachou, Christos Doulkeridis, and Kjetil Nørvåg. *AGiDS: A Grid-Based Strategy for Distributed Skyline Query Processing*, pages 12–23. Springer, 2009.

[36] M. A. Siddique, H. Tian, and Y. Morimoto. k-dominant skyline query computation in mapreduce environment. *IEICE Transactions on Information and Systems*, pages 1745–1361, 2015.

[37] Md. Anisuzzaman Siddique and Yasuhiko Morimoto. Efficient selection of various k-objects for a keyword query based on mapreduce skyline algorithm. In *Proceedings of the 9th International Workshop on Databases in Networked Information Systems - Volume 8381*, DNIS 2014, pages 40–52, New York, NY, USA, 2014. Springer-Verlag New York, Inc.

[38] Md. Anisuzzaman Siddique, Hao Tian, and Yasuhiko Morimoto. Distributed skyline computation of vertically splitted databases by using mapreduce. In *Database Systems for Advanced Applications: 19th International Conference, DASFAA 2014, International Workshops: BDMA, DaMEN, SIM, UnCrowd; Bali, Indonesia, April 21–24, 2014, Revised Selected Papers*, pages 33–45, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[39] Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-based representative skyline. In *2009 IEEE 25th International Conference on Data Engineering*, pages 892–903, March 2009.

[40] Y. Tao, W. Lin, and X. XIAO. Minimal mapreduce algorithm. In *Proceedings of SIGMOD*, pages 529–540, 2013.

[41] H. Tian, M. A. Siddique, and Yasuhiko Morimoto. An efficient processing of k-dominant skyline query in mapreduce. In *Proceedings of ACM International Workshop on Bringing the Value of Big Data to Users (Data4U)*, pages 29–35, 2014.

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

[43] A. Vlachou, C. Doulkeridis, Y. Kotidis, and M. Vazirgiannis. Skypeer: Efficient subspace skyline computation over distributed data. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 416–425, April 2007.

[44] S. Wang, B. C. Ooi, A. K. H. Tung, and L. Xu. Efficient skyline query processing on peer-to-peer networks. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 1126–1135, April 2007.

[45] Ross Williams. A painless guide to CRC error detection algorithms. In *ftp.rocksoft.com/papers/crc_v3.txt*. 1996.

[46] Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, pages 160–164, 1982.

[47] 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.

[48] Boliang Zhang, Shuigeng Zhou, and Jihong Guan. Adapting skyline computation to the mapreduce framework: Algorithms and experiments. In *Proceedings of DASFAA'11*, pages 403–414. Springer-Verlag, 2011.

List of Referred Publications

Referred Journals

J-1 **Asif Zaman**, Md Anisuzzaman Siddique, Annisa and Yasuhiko Morimoto, *"Finding Key Persons on Social Media by Using MapReduce Skyline"*, International Journal of Networking and Computing (IJNC), Vol 7, Issue 1, Pages 86-104, January 2017

J-2 **Asif Zaman**, Md Anisuzzaman Siddique, Annisa and Yasuhiko Morimoto, "*Secure k-objects selection for a keyword query based on MapReduce skyline algorithm*", International Journal of Computational Science and Engineering (IJCSE), In Press. Inderscience Publishers, Switzerland.

Referred International Conferences

C-1 **Asif Zaman**, Md. Anisuzzaman Siddique, Annisa, and Yasuhiko Morimoto *"Selecting Key Person of Social Network Using Skyline Query in MapReduce Framework"* Proceedings of the Third International Symposium on Computing and Networking (CANDAR,15), pp:213-219, Sapporo, Hokkaido, Japan December 8-11, 2015 ,doi: 10.1109/CANDAR.2015.84

C-2 **Asif Zaman**, Md. Anisuzzaman Siddique, Annisa, and Yasuhiko Morimoto *"Secure Computation of Skyline Query in Mapreduce"*, Proceedings of 12 International Conference Advanced data Mining and Applications, ADMA 2016, pp 345-360 December 12-15, Gold Coast, QLD, Australia December 12-15, 2016. doi: 10.1007/978-3-319-49586-6_23. Lecture Notes in Computer Science (LNCS) 10086, Springer, Switzerland.

Other Publications **(not in dissertation)**

Referred Journals

J-3 Md. Anisuzzaman Siddique, **Asif Zaman** and Yasuhiko Morimoto *"Efficient Selection of Representative Combinations of Objects from Large Database"* International Journal on Advances in Software, issn 1942-2628 vol. 8, no. 3 & 4, pages 484 - 490, year 2015

J-4 Annisa, **Asif Zaman** and Yasuhiko Morimoto *"Area Skyline Query for Selecting Good Locations in a Map"*, Journal of Information Processing, IPSJ - Information Processing Society of Japan, Vol:24, Issue 5, Page 946-955, November 2016

J-5 Annisa, **Asif Zaman** and Yasuhiko Morimoto *"Reverse Area Skyline in a Map"*, International Journal of Advanced Computer Science and Applications vol. 8, no. 2, pp. 333-343, March 2017. (ESCI indexed)

Referred International Conferences

C-3 Md. Anisuzzaman Siddique, **Asif Zaman**, and Yasuhiko Morimoto, *"Skyband-Set for Answering Top-k Set Queries of Any Users"*, Proceedings of the 10th International Workshop on Databases in Networked Information Systems (DNIS 2015), Japan, March, 2015, Lecture Notes in Computer Science (LNCS) 8999, pp. 41-55, Springer, Switzerland 2015

C-4 Md. Anisuzzaman Siddique, **Asif Zaman**, Yasuhiko Morimoto *"Skyline Objectset: Efficient Selection of Non-dominate Sets from Database"*, Proceedings of The Seventh International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA 2015), held in Rome, Italy - May 24 - 29, 2015

C-5 Annisa, Md. Anisuzzaman Siddique, **Asif Zaman**, and Yasuhiko Morimoto, *"A Method for Selecting Desirable Unfixed Shape Areas from Integrated Geographic Information System"*, Proceedings of the 4th International Congress on Advanced Applied Informatics (AAI 2015), pp. 195-200, Japan, July, 2015.