Computing and Informatics, Vol. 33, 2014, 831–856

PRIVACY AWARE PARALLEL COMPUTATION OF SKYLINE SETS QUERIES FROM DISTRIBUTED DATABASES

Mohammad Shamsul Arefin

Graduate School of Engineering
Hiroshima University
Kagamiyama 1-7-1, Higashi-Hiroshima 739-8521, Japan
&
Department of Computer Science and Engineering
Chittagong University of Engineering and Technology
Chittagong-4349, Bangladesh
e-mail: sarefin_406@yahoo.com

Yasuhiko Morimoto

Graduate School of Engineering
Hiroshima University
Higashi-Hiroshima 739-8521, Japan
e-mail: morimoto@mis.hiroshima-u.ac.jp

Abstract. A skyline query finds objects that are not dominated by another object from a given set of objects. Skyline queries help us to filter unnecessary information efficiently and provide us clues for various decision making tasks. However, we cannot use skyline queries in privacy aware environment, since we have to hide individual's records values even though there is no ID information. Therefore, we considered skyline sets queries. The skyline set query returns skyline sets from all possible sets, each of which is composed of some objects in a database. With the growth of network infrastructure data are stored in distributed databases. In this paper, we expand the idea to compute skyline sets queries in parallel fashion from distributed databases without disclosing individual records to others. The proposed method utilizes an agent-based parallel computing framework that can efficiently compute skyline sets queries and can solve the privacy problems of skyline queries in distributed environment. The computation of skyline sets is performed simultaneously in all databases which increases parallelism and reduces the computation time.

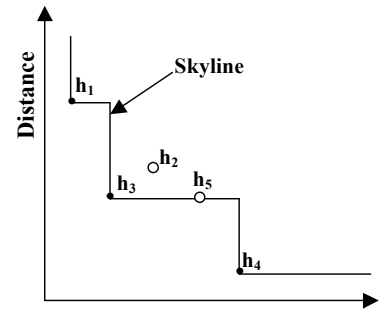
Keywords: Skyline sets, convex skyline, parallel computation, agent-based computation, compromisable situations

1 INTRODUCTION

Given a k-dimensional database DB, a skyline query retrieves a set of skyline objects, each of which is not dominated by another object. An object p is said to dominate another object q if p is not worse than q in any of the k dimensions and p is better than q in at least one of the k dimensions. Figure 1 shows a typical example of skyline. The table in Figure 1 is a list of five hotels, each of which contains two numerical attributes – "Price" and "Distance". In the list, h_2 and h_5 are dominated by h_3, while others are not dominated by any other hotel. Therefore, the skyline of the list is $\{h_1, h_3, h_4\}$. Such skyline results are important for users to take effective decisions over complex data having many conflicting criteria.

ID	Price	Distance
h_1	3	8
h_2	5	4
h_3	4	3
h_4	9	2
h_5	7	3

(a) Hotels



(b) Skyline

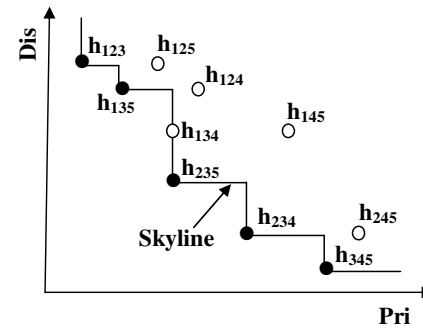
Figure 1. Skyline example

A number of efficient algorithms for computing skyline from a sole database have been reported in the literature [1, 2, 3, 4, 5, 6]. With rapid growth of computer networks, parallel and distributed skyline query processing has attracted attention [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19].

Recently, we need to be aware of individual's privacy. In privacy aware environments, in general, we have to hide individual record values even though there is no ID information. For example, in case of employees database of an organization, we are not allowed to disclose the salaries and experiences of the employees to others. The main limitation of all the previous works of the skyline query is that they have to disclose exact values of each record to others. In addition, current skyline query algorithms are not robust against data with outliers and are not stable with update operation. Moreover, conventional skyline query algorithms do not provide any facility for group choice, although sometimes users are interested in group of objects instead of individual objects.

ID	Pri	Dis	ID	Pri	Dis
$\mathbf{h_{123}}$	12	15	$\mathbf{h_{145}}$	19	13
$\mathbf{h_{124}}$	17	14	$\mathbf{h_{234}}$	18	9
$\mathbf{h_{125}}$	15	15	$\mathbf{h_{235}}$	16	10
$\mathbf{h_{134}}$	16	13	$\mathbf{h_{245}}$	21	9
$\mathbf{h_{135}}$	14	14	$\mathbf{h_{345}}$	20	8

(a) Sets of 3 hotels



(b) Skyline of 3 hotels

Figure 2. Skyline of 3-set

To overcome the above limitations of conventional skyline queries, we have introduced skyline sets queries [22, 23]. Let s be the number of objects in each set and n is the number of objects in the dataset. The number of sets in the database amounts to ${}_nC_s$. We proposed an efficient algorithm to compute skyline of ${}_nC_s$ sets.

Figure 2 is a list of 3-sets, in which all of the combinations of three hotels are listed. In Figure 2, "ID" denotes a set of three hotels and the attributes "Pri" and "Dis" represent the sums of price and distance of three hotels of Figure 1, respectively. In the example h_{123} denotes a set of three hotels $\{h_1, h_2, h_3\}$. "Pri" and "Dis" of h_{123} are the sums of the "Price" and "Distance" of hotels in the set, respectively. The skyline of the combinations of three hotels are $\{h_{123}, h_{135}, h_{235}, h_{234}, h_{345}\}$ as shown Figure 2. If a tourist wants to know the price of the cheapest hotel, she/he can easily estimate that the price of the cheapest hotel is around 4 from the value of the cheapest set h_{123}. Similarly, if one prefers cheaper and closer one, she/he can predict the best price and distance will be around 5.33 and 3.33, respectively from the value of h_{235}. Based on such skyline sets, the tourist can make an expenditure plan for her/his accommodations. Note that in our skyline sets queries, each attribute value of each record is not disclosed to preserve the privacy of individuals. Instead, we disclosed aggregated values. Also observe that in the example of 3-sets computation in Figure 2, we have used "sum" aggregation function. We can also employ other aggregation functions such as "average" or "mean". However, in either case, the result will be the same as the result of "sum" aggregation function. For example, after computing average price and distance for all combinations of three hotels, the result of skyline 3-sets is $\{h_{123}, h_{135}, h_{235}, h_{234}, h_{345}\}$ that is the same as the result obtained previously by using "sum" aggregation function.

Our work in [22] considered skyline sets queries from a centralized database. Later in [23], we introduced a framework for skyline sets computation from distributed databases. In [23], we have considered simple pipeline execution. Though it can be used in distributed databases, the computation time becomes slower if the number of databases increases. Another limitation of this work is that in some situ-

ID	a_1	a_2
o_1	5	5
o_2	1	6
o_3	6	4
o_4	9	7
o_5	6	8

DB_1

ID	a_1	a_2
o_6	1	3
o_7	7	2
o_8	7	9
o_9	4	5
o_{10}	6	7

DB_2

ID	a_1	a_2
o_{11}	4	2
o_{12}	8	4
o_{13}	4	1
o_{14}	8	7
o_{15}	1	5

DB_3

Figure 3. Three databases with the same schema

ations there are possibilities of disclosure of record values from some of aggregated values.

ID	a_1	a_2
R_1	3	14
R_2	15	5
R_3	6	9
R_4	9	6

Table 1. Results of skyline sets queries for $s = 3$ from the databases of Figure 3

In this paper, we proposed an efficient agent-based parallel computation framework, where there is almost no performance degradation with the increase in the number of databases. Moreover, in the proposed approach we solved the statistical compromise problem, which is important in privacy aware environment.

1.1 Motivating Example

Our previous works in [22, 23] can compute skyline sets queries without disclosing individual record values. However, there are several situations where users can infer the individual record values from the skyline sets. Let us consider the three databases with the same schema (ID, a_1, a_2) as shown in Figure 3. In this example, we assume that value domain of both a_1 and a_2 is set to $[1 \dots 10]$.

Skyline sets queries with $s = 3$ return the results as shown in Table 1 as the skyline of 3-sets. Notice that the aggregated values in Table 1 are disclosed to the public including database owners.

From the a_1 of R_1, any user can easily find that individual record value in a_1 is 1 because the minimum value in a_1 is 1. Therefore, the owner of DB_1 can find there are two records whose a_1 is 1 in other databases. Similarly, the owner of DB_2 and DB_3 can find the fact.

From the value of a_2 of R_2, one can find that one of the three records contains 1 in a_2 since aggregated value of 3 records is 5. Therefore, one can infer that the remaining two records have values 1 and 3 or 2 and 2 in a_2. In this case, the owner of DB_3 happens to know that one of others has 2 in a_2 and no other has 1 in a_2 since DB_3 has 1 and 2.

In this paper, we provide a framework for detecting such compromising skyline s-sets and propose a protection mechanism for such s-sets. We also introduce an efficient agent-based parallel computation framework that improves the overall computation performance significantly.

The rest of this paper is organized as follows. Section 2 gives the preliminaries of the problems. In Sections 3, we detail the privacy aware parallel execution of skyline sets queries. Section 4 presents the experimental results. Section 5 provides a brief review of related works on skyline queries. Finally, we conclude and sketch future research directions in Section 6.

2 PRELIMINARIES

We consider a database DB having k attributes and n objects. Let $a_1, a_2, \ldots, a_k$ be the k attributes of DB. Without loss of generality, we assume that smaller values in each attribute are better and each attribute contains positive integer values.

2.1 Skyline Queries

Let p and q be objects in DB. Let $p.a_l$ and $q.a_l$ be the l^{th} attribute values of p and q, respectively, where $1 \leq l \leq k$. An object p is said to dominate another object q, if $p.a_l \leq q.a_l$ for all the k attributes a_l, $(1 \leq l \leq k)$ and $p.a_j < q.a_j$ on at least one attribute a_j, $(1 \leq j \leq k)$. The skyline is a set of objects which are not dominated by any other object in DB.

2.2 Skyline Sets Problem

Let $|S| =_n C_s = \frac{n!}{s!(n-s)!}$ be the number of s-sets that can be composed from n objects. We assume a virtual database of S on the k dimensional space of DB. Each object of the database is an s-set whose value of each attribute (dimension) is the sum of s values of corresponding s objects. An s-set $p \in S$ is said to dominate another s-set $q \in S$, denoted as $p \leq q$, if $p.a_l \leq q.a_l$, $1 \leq l \leq k$ for all k attributes and $p.a_j < q.a_j$, $1 \leq j \leq k$ for at least one attribute. We call such p dominant s-set and q dominated s-set between p and q.

An s-set $p \in S$ is said to be a skyline s-set if p is not dominated by any other s-set in S.

2.3 Convex Skyline Sets

Each object in S is a point in k-dimensional vector space. Convex hull for the set of S points is the minimum convex solid that encloses all of the objects of S. The dotted line polygon of Figure 4 is an example of convex hull in two-dimensional space. In Figure 4, O_1 and O_4 are the objects that have the minimum values of attribute in a_1 and a_2, respectively. Notice that such objects must be in the convex

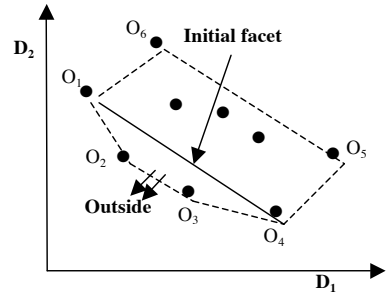


Figure 4. Convex hull and convex skyline

hull. We call the line between O_1 and O_4 "the initial facet". Among all objects in the convex hull, objects that lie outside of the initial facet are skyline objects and we call such objects "convex skyline objects". In k-dimensional space, we compute such initial hyperplane surrounded by k objects as the initial facet. Then, we compute convex skyline objects that lie in the convex hull and outside the initial facet.

The definition of convex skyline sets problem can be simplified as follows:

Given a natural number s, find all s-sets which lay in both the convex hull and the skyline of S.

2.4 Algorithm for Computing Convex Skyline Sets

If we compute all of the s-sets from the original database and make a dataset containing $|S|$ records, the problem can be solved by conventional skyline query algorithms. However, $|S|$ is unacceptably large when the original database size is large. Therefore, in [22] we proposed an efficient alternative.

Each s-set in S can be represented as a k-dimensional point $x = (x_1, x_2, \ldots, x_k)$ where x_i, $(1 \leq i \leq k)$ is the sum of the i^{th} attribute value of the s objects in DB. "Touching oracle" function proposed in [22] is a method to compute an s-set on the convex hull without generating S. It computes the tangent object of the convex hull of S and a $(k-1)$-dimensional hyperplane directly from the original n records in DB.

2.4.1 Touching Oracle

Assume there is a hyperplane whose normal vector is Θ. In order to find the tangent point with the hyperplane and the convex hull without precomputing S, we compute (Θ, o), i.e., inner products of the normal vector and each object o in the database. We choose s objects whose inner products are the top s. These top s objects compose the s-set of the tangent point. Consider the hotel example as shown in Figure 1. There are five records in the original database DB as in Figure 1(a). Each of the five records is represented as a two-dimensional point, which we call an atomic point.

o	$(\Theta_1, \mathbf{o})$	$(\Theta_2, \mathbf{o})$	$(\Theta_{1,2}, \mathbf{o})$
$\mathbf{h_1}$	-3	-8	-85
$\mathbf{h_2}$	-5	-4	-67
$\mathbf{h_3}$	-4	-3	-52
$\mathbf{h_4}$	-9	-2	-79
$\mathbf{h_5}$	-7	-3	-73

Table 2. Inner product with tangent lines

Now consider the line whose normal vector is $\Theta_1 = (-1, 0)$. In order to find the tangent point corresponding to the line with Θ_1, we compute inner products of the normal vector and each of the five atomic points as shown in the second column of Table 2. Then, we choose the top three inner products, i.e., $\{h_1, h_2, h_3\}$ if $s = 3$. These top three inner products compose the tangent point (12, 15), which is the 3-set, h_{123}. Similarly, for a line with $\Theta_2 = (0, -1)$, we can find $\{h_3, h_4, h_5\}$ as the top three. Those three points compose the tangent point (20, 8).

As mentioned above, we can compute a tangent point, which is a point on the convex hull, by giving the normal vector of a tangent line. In k-dimensional case, we can find a tangent point with a tangent $(k-1)$-dimensional hyperplane by giving the normal vector of the tangent $(k-1)$-dimensional hyperplane.

The touching oracle function chooses the top-s points from n atomic points in *DB*. Since s is a negligible small constant compared with n, we can compute the tangent point by scanning n atomic points only once, which is $O(n)$.

2.4.2 Convex Hull Search

First, we compute initial k tangent objects that can be computed by touching oracle with initial k vectors $\Theta_x = (\theta_1, \theta_2, \ldots, \theta_k)$, where $\theta_i = -1$ if $i = x$, otherwise $\theta_i = 0$ for each $x = 1, \ldots, k$. Note that those k initial tangent objects are on the horizon of the initial facet ($(k-1)$-dimensional hyperplane). Convex skyline s-sets are objects which lie outside of the initial facet and are in the convex hull.

Next, we compute the normal vector of the initial facet. Using the computed normal vector, we try to find new tangent point. The new tangent point expands the initial facet into k facets. We recursively compute the touching oracle for each of the expanded facets until we can find a new tangent object outside the facet.

From Table 2, for example, we have obtained two initial tangent points $p_1 = (12, 15)$ and $p_2 = (20, 8)$ with normal vectors $\theta_1 = (-1, 0)$ and $\theta_2 = (0, -1)$, respectively. These two initial tangent points construct the initial facet. Using the facet containing the two initial points, we can compute the normal vector of the facet as $\theta_{1,2} = (-(15-8), (12-20)) = (-7, -8)$, which directs outside of the facet. Using this normal vector, we can find new tangent point h_{235}, which is (16, 10). The new tangent point expands the initial facet into two facets, which are the facet surrounded by $p_1 = (12, 15)$ and (16, 10) and the facet surrounded by (16, 10) and $p_2 = (20, 8)$.

We can apply this operation for higher k-dimensional space analogically using the concept of [26].

2.5 Advantages of Skyline Sets

As we mentioned, in a privacy aware environment we cannot disclose individual record values even if there is no ID information. In such an environment, we cannot use conventional skyline query. Our skyline sets query can be a promising alternative since we do not have to disclose values of each record in a skyline analysis.

In addition to the advantage of the privacy issue, the skyline sets query has another significant advantage against conventional skyline query, which is "robustness".

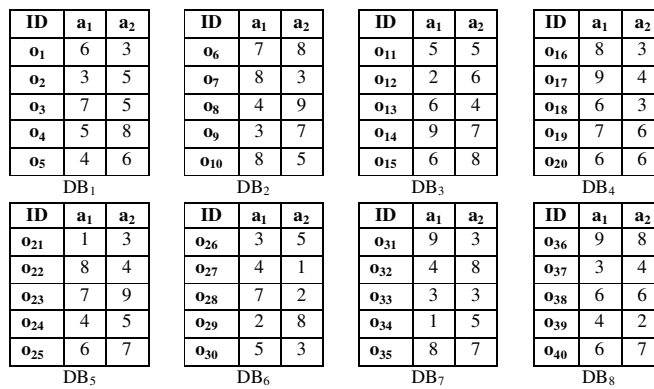


Figure 5. Distributed database example

For example, if there are few outliers in a database, conventional skyline queries tend to retrieve outliers as skyline objects. Such retrieved outliers often hide some important skyline points, which is significant loss of information. Our proposed skyline sets queries can reduce the effect of outliers.

If there is an outlier whose value is $(1, 1)$ in the hotel example of Figure 1, any conventional skyline query retrieves this outlier record as a skyline. If one wants to know the cheapest hotel or the nearest hotel, she/he cannot find any clue from conventional skyline query. On the other hand, the skyline set query can provide better results for her/him. For example, if $s = 3$, she/he can find the price of the cheapest 3-set is 8, which is $3 + 4 +$ outlier, and the distance of the nearest 3-set is 6, which is $2 + 3 +$ outlier. Note that in an actual application the number of records n in a database is very large and s is also larger than three. In such an actual situation, the aggregated values of the skyline set query can soften the effect of the noises, though these aggregated values include the noise of outliers.

Since skyline takes time to compute, we usually use precomputed skyline results to answer a query quickly. If a record in a database is updated, we have to recompute

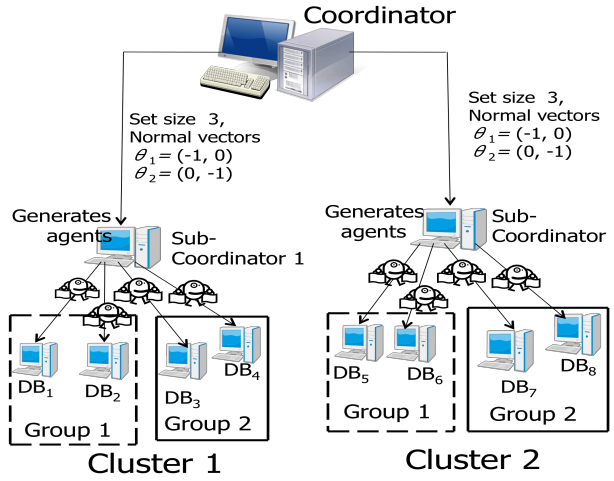


Figure 6. Example of divide-and-conquer computation

the skyline results. However, if records in a database are frequently updated, we cannot prepare the precomputed results in time. Assume that the cheapest hotel $h_1 = (3, 8)$ in the example of Figure 1 is booked and is deleted. The precomputed skyline $\{h_1, h_3, h_4\}$ is meaningless for users who are interested in the cheapest price unless it is recomputed. This problem has been an important research issue of skyline query. Skyline set query can soften the effects of the updates that may exist until the next recomputation. For example, the cheapest 3-set h_{123} in Figure 2 has valuable information for users who want to know the price of the cheapest hotel even if $h_1 = (3, 8)$ of Figure 1 is no longer existing.

3 SECURE PARALLEL COMPUTATION OF SKYLINE SETS

3.1 Computing Convex Skyline Sets

We assume that there are m databases in a network. Let $DB_1, DB_2, \ldots, DB_m$ be the databases. Each database has a view table whose schema has the following columns: ID, a_1, a_2, …, a_k, where ID is the primary key attribute and a_i $(i = 1, \ldots, k)$ are k-dimensional numerical attributes. Assume that we have to compute skyline sets for the union of m such databases in such a way that the privacy of individual is preserved. Figure 5 is an example of a distributed database that consists of eight databases, DB_1, DB_2, …, and DB_8, each of which lies in a different server.

3.2 Agent-based Parallel Computation

We assume there is a coordinator who is responsible for performing the convex hull search, which is mentioned in Section 2.4. The coordinator computes the touching oracle function, which is to find the top-s inner product values from distributed databases, by the divide-and-conquer strategy.

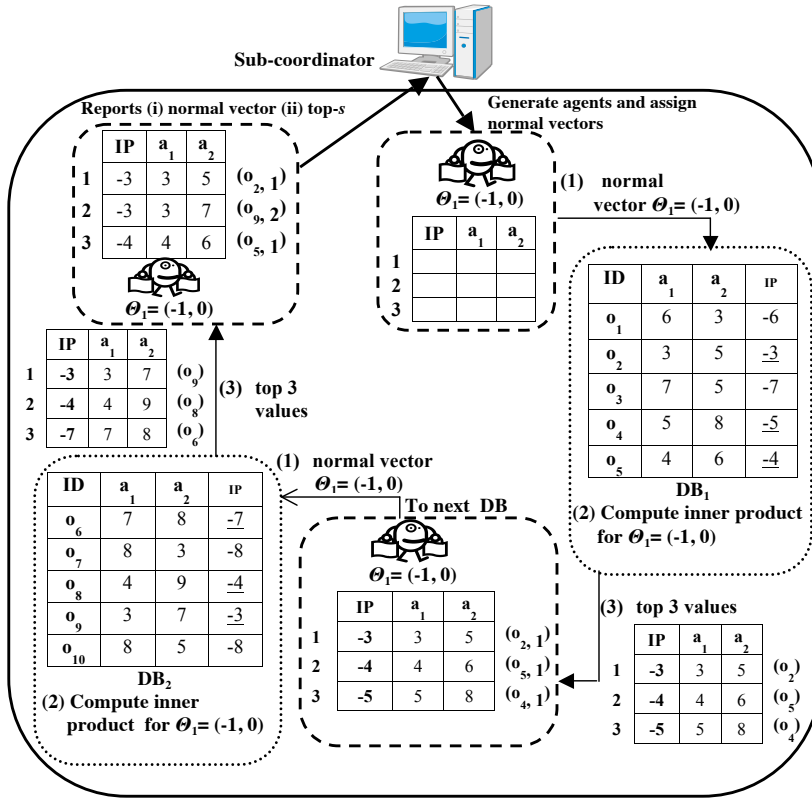


Figure 7. Computation in group one of cluster one with $\Theta_1 = (-1, 0)$

The coordinator divides the distributed databases into several clusters and creates sub-coordinators for each cluster. For each cluster, the sub-coordinator computes the "local" top-s among the databases in the corresponding cluster. After computing all "local" top-s, the coordinator merges all the "local" top-s and finds "global" top-s. During the process, agents are used to preserve privacy of all "local" databases. Note that all the "local" computations are performed simultaneously. Now, consider the secure computation of skyline 3-set query from the distributed databases of Figure 5. For each cluster, the coordinator asks the sub-coordinator to compute touching oracles for the two initial normal vectors, i.e., $(-1, 0)$ and $(0, -1)$.

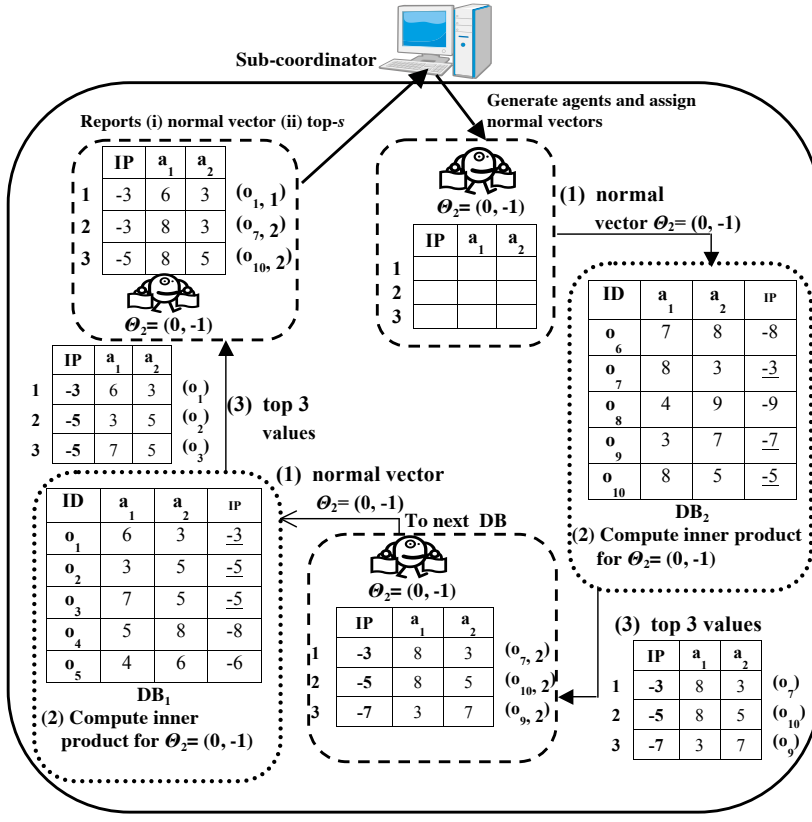


Figure 8. Computation in group one of cluster one with $\Theta_2 = (0, -1)$

3.2.1 Computation in Each Cluster

In order to minimize idle time, the sub-coordinator divides databases into several groups. In general, the number of groups is the same as the number of different normal vectors, which are in the process. However, if the number of databases in a cluster is less than the number of normal vectors, we set the number of groups to the number of databases. For example, if we are processing the two initial normal vectors, databases of the cluster are divided into two groups as in Figure 6.

For each group, the sub-coordinator creates two agents one of which is for $(-1, 0)$ and the other is for $(0, -1)$. Each agent has a normal vector and a priority queue, also known as "heap data structure" that keeps the top-3 inner product values and their corresponding record values.

Figure 7 shows the computation process of the agent with normal vector $\Theta_1 = (-1, 0)$ in group 1 of cluster 1. When an agent arrives at a database of a group, it sends the normal vector of the database. Next, the database computes inner product

for each record of the database. Finally, the database pushes the local top-3 records along with inner product values to the agent. Note that during this computation the database cannot see the contents of the priority queue of the agent.

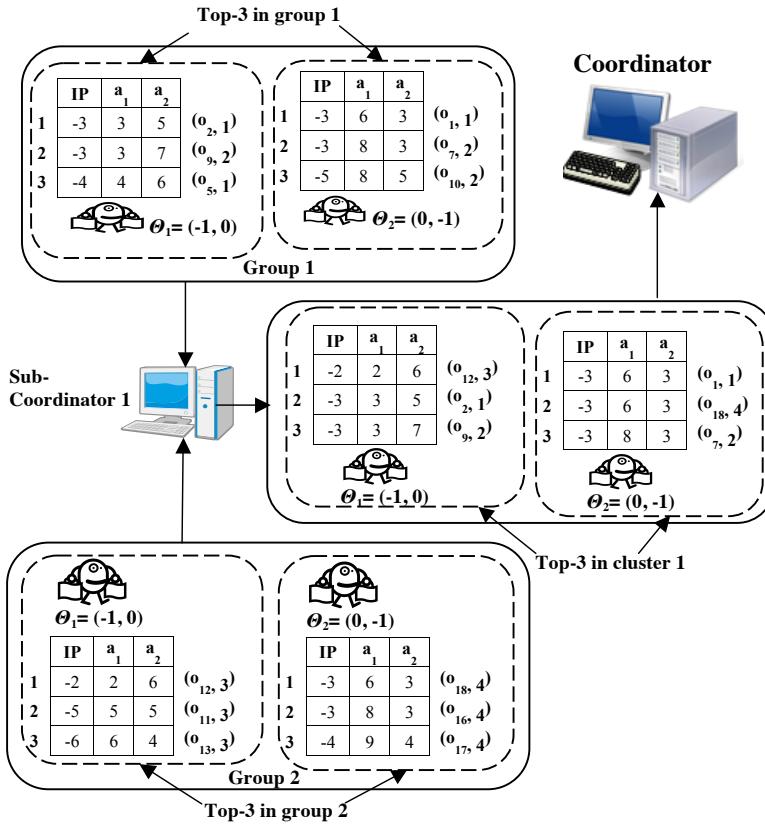


Figure 9. Merging process at cluster one

In the example of Figure 7, it is observed that DB_1 pushes three triplets, i.e., inner product value (IP), a_1 value, and a_2 value, $(IP, a_1, a_2) = ((-3, 3, 5), (-4, 4, 6), (-5, 5, 8))$ to the agent. Next, the agent goes to DB_2. DB_2 pushes $(IP, a_1, a_2) = ((-3, 3, 7), (-4, 4, 9), (-7, 7, 8))$ to the agent. After visiting all databases in the cluster, agent with normal vector $\Theta_1 = (-1, 0)$ contains $(IP, a_1, a_2) = ((-3, 3, 5), (-3, 3, 7), (-4, 4, 6))$ in its priority queue and returns back to the sub-coordinator. During the top-3 computation the agent keeps track about the owner of each of the three objects. For example, we can see that two objects are from DB_1 and one object from DB_2 in Figure 7.

Figure 8 shows similar computation in group 1 of cluster 1 with normal vector $\Theta_2 = (0, -1)$. Here, in order to minimize idle time, the agent travels from

Algorithm 1 ComputationWithinCluster

1: **Input:** Normal vectors θs, set size s
2: **Output:** Local top-s for each normal vector
3: **begin**
4: Let $(\theta_1, \theta_2, \ldots, \theta_v)$ be the given v normal vectors, $(DB_1, DB_2, \ldots, DB_z)$ be z databases in the cluster
5: **if** $z < v$ **then**
6: Create z groups
7: **else**
8: Create v groups
9: **end if**
10: **for** each i ($i = 1$ to z) **do**
11: Assign DB_i to group (i % v)
12: **end for**
13: **for** each group **do**
14: Create agents $ag(\theta_t)$, $(1 \leq t \leq v)$ // Each $ag(\theta_t)$ is an agent with θ_t
15: With the help of $ag(\theta_t)$, compute top-s from the DBs of the group
16: **end for**
17: Send top-s of θ_t, $(1 \leq t \leq v)$ for each group to the sub-Coordinator and compute top-s of θ_t in the cluster at the sub-coordinator
18: **end**

DB_2 and goes to DB_1. After the computation, the agent contains $(IP, a_1, a_2) = ((-3, 6, 3), (-3, 8, 3), (-5, 8, 5))$ in its priority queue and goes to the sub-coordinator and reports the results.

During these processes, the sub-coordinator computes the local top-3 in group 2 simultaneously. After the agent-based computation in two groups, the sub-coordinator merges the local top-3 priority queues for each normal vector. Figure 9 shows the merge process of cluster 1.

Algorithm 1 shows the procedure for local top-s computation within a cluster. First, it creates necessary groups and divides the databases among the groups (lines 5–12). Next, it computes top-s values for each normal vector from the groups (lines 13–16). Finally, it calculates local top-s for the normal vectors in the cluster (line 17).

3.2.2 Global Merging

After the computation of local top-s for a normal vector in a cluster, the result is sent to the coordinator for global merging. In global merging, local top-s results from the clusters are merged to obtain global top-s results. For example, local top-3 results corresponding to each normal vector from two clusters of Figure 6 are merged into global top-3 as shown in Figure 10. These merge processes are carried out among the agents so that the coordinator does not see the individual

record values of the agents. The agent then returns the aggregated values to the coordinator.

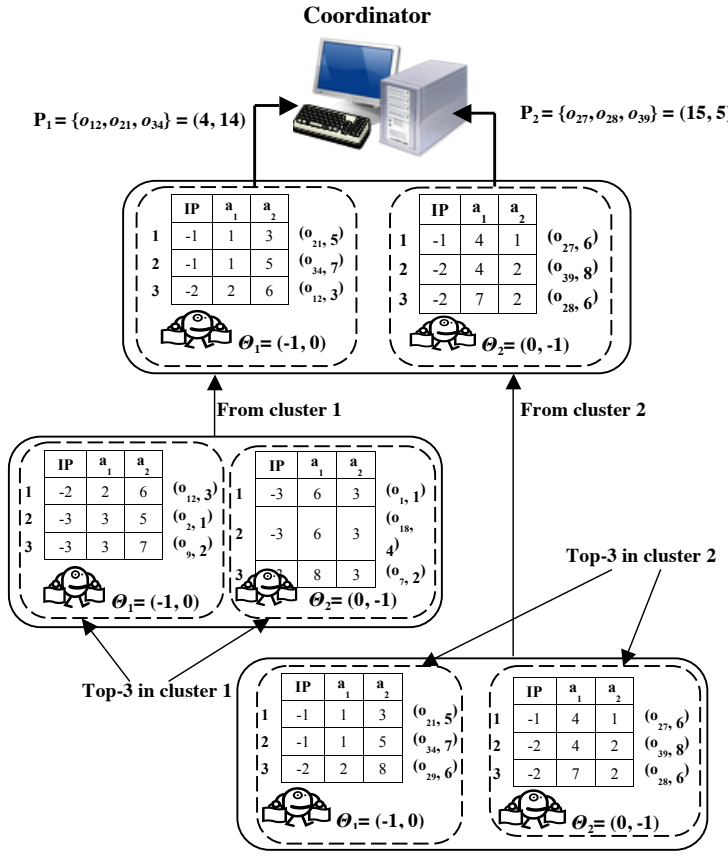


Figure 10. Merging process at the coordinator

From the example of Figure 5, we get that 3-set corresponding to normal vector $(-1, 0)$ is $\{o_{12}, o_{21}, o_{34}\}$, whose coordinate values in the two-dimensional space are $(4, 14)$ and that the 3-set corresponding to normal vector $(0, -1)$ is $\{o_{27}, o_{28}, o_{39}\}$ whose coordinate values are $(15, 5)$.

The global merging procedure is given in Algorithm 2. The algorithm first computes global top-s from local top-s of the clusters (lines 4–6). Next, it calculates aggregated values of attributes of top-s and returns the result to the coordinator (lines 7–10).

Algorithm 2 GlobalMerging

Input: Local top-s of θ_t, $(1 \leq t \leq v)$ for each cluster
Output: Aggregated values correspond to each θ_t, $(1 \leq t \leq v)$
begin
for each t ($t = 1$ to v) **do**
  Compute "global" top-s of θ_t from "local" top-s for each cluster
end for
for each θ_t, $(1 \leq t \leq v)$ **do**
  Compute the summation of values of each dimension of top-s objects
  return the aggregated values to the coordinator
end for
end

3.2.3 Facet Expansion

After receiving all surrounding points of a facet, the coordinator computes the normal vector of the facet by using the surrounding points. In the example, $P_1 = (4, 14)$, which is found by normal vector $\theta_1 = (-1, 0)$ and $P_2 = (15, 5)$, which is found by normal vector $\theta_2 = (0, -1)$, are two surrounding points of the initial facet (line segment between P_1 and P_2). The coordinator computes the normal vector $\theta_{1,2} = (-9, -11) = (-(14-5), (4-15))$ from the facet as shown in Figure 11. Then, the normal vector $\theta_{1,2} = (-9, -11)$ is sent to the sub-coordinator of each cluster and the agent-based parallel touching oracle finds $P_{1,2} = (9, 6)$, which is composed of $\{o_{21}, o_{27}, o_{39}\}$. This point expands the initial facet (line segment between P_1 and P_2) into two facets, which are the line segment between P_1 and $P_{1,2}$ and the line segment between $P_{1,2}$ and P_2 as shown in Figure 11.

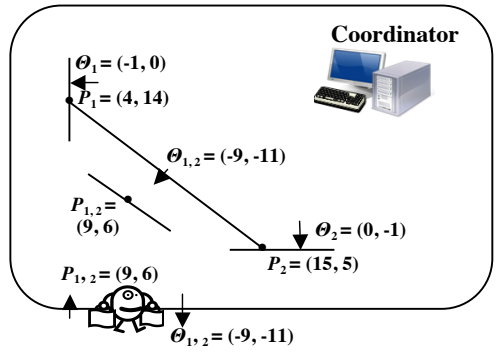


Figure 11. Facet expansion

We recursively compute tangent points for each of the expanded facets. If we find a new point outside the facet, we expand the facet further. We continually

adopt the recursive operation while we can find new tangent point outside the facet. Finally, we can find all the points of convex skyline s-sets.

3.3 Compromisable Situations in Skyline Sets Queries

Let us assume that domains of the numerical attributes are specified. Let min_i and max_i be the minimum and maximum values of an attribute a_i, $(i = 1, \ldots, k)$. Let $value_i$ is the aggregated value of a_i in an s-set. We can say that a skyline s-set is a compromised s-set if it contains at least one aggregated value $value_i$ such that we can find exact a_i's value of a member record of an s-set from $value_i$. Formally, we can say that a skyline s-set is compromised if $(value_i - s * min_i) < s$ or $(s * max_i - value_i) < s$, in an attribute a_i, $(i = 1, \ldots, k)$. For example, assume that $s = 3$ and $min_1 = 1$. If we have an s-set whose $value_1$ is 4, one can find that three member values in a_1 are 1, 1, and 2. Note that this example satisfies the condition $(value_i - s * min_i) < s$. Similarly, if $s = 4$, $max_1 = 5$, $value_1 = 17$, we can find that there is at least one record whose a_i value is 5 in the s-set.

ID	a_1	a_2
o_1	5	5
o_2	1	9
o_3	6	4
o_4	9	7
o_5	1	8

DB_1

ID	a_1	a_2
o_6	1	8
o_7	8	1
o_8	9	2
o_9	4	5
o_{10}	8	5

DB_2

ID	a_1	a_2
o_{11}	8	2
o_{12}	8	4
o_{13}	4	6
o_{14}	8	7
o_{15}	2	5

DB_3

Figure 12. Example of three databases having compromisable 3-sets

ID	a_1	a_2
Q_1	3	25
Q_2	25	5
Q_3	14	11
Q_4	18	8
Q_5	4	21

Table 3. Some skyline 3-sets in the databases of Figure 12

Now, consider an example of three databases as shown Figure 12. We assume that domain of a_1 and a_2 are $[1 \ldots 9]$, i.e., $min_1 = min_2 = 1$ and $max_1 = max_2 = 9$. Table 3 shows partial results of skyline sets queries with $s = 3$. According to the definition of compromised s-sets, we can find that skyline 3-sets $Q_1 = (3, 25)$, $Q_2 = (25, 5)$, and $Q_5 = (4, 21)$ are compromised 3-sets. However, Q_3 and Q_4 are not compromised 3-sets.

We considered a perturbation method to prevent the compromised skyline s-sets. The coordinator performs the detection of compromised s-sets and perturbation of

Algorithm 3 Perturbation

```
1: Input: Skyline s-sets
2: Output: Perturbed s-sets
3: begin
4: for each skyline s-set do
5:   for each attribute a_i do
6:     if ((value_i − s ∗ min_i) < s) then
7:       return value_i = (s + s ∗ min_i)
8:     else if ((s ∗ max_i − value_i) < s) then
9:       return value_i = (s ∗ max_i − s)
10:    else
11:      return value_i
12:    end if
13:  end for
14: end for
15: end
```

them using Algorithm 3. Algorithm 3 first checks whether there is any compromised value (line 6 and line 8) in the corresponding skyline s-set. If the algorithm detects a compromised value, it replaces the value with a new value (line 7 and line 9).

Now, consider compromised 3-sets Q_1, Q_2, and Q_5 of Table 3. From Q_1, we can see that it has a value 3 at attribute a_1 that satisfies line 6 of Algorithm 3. So, Algorithm 3 replaces the value 3 with a value $(3 + 3 * 1) = 6$ (line 7). Like this, Algorithm 3 replaces any value $value_i$ that satisfies the condition of line 6 with a value equal to $(s + s * min_i)$.

We can further observe that the value 25 of Q_1 at attribute a_2 satisfies line 8 of Algorithm 3. So, the algorithm replaces the value 25 with a value $(3 * 9 - 3) = 24$ (line 9). Algorithm 3 performs the replacement of any such value with a value equal to $(s * max_i s)$. After such replacements Q_1 has values 6 and 24 in its attributes a_1 and a_2, respectively. Note that after replacement, from Q_1 no one can infer a member value exactly. Similarly, after perturbation, Q_2 and Q_5 are modified to (24, 6) and (6, 21), respectively and no one can exactly identify a member value from Q_2 or Q_5.

If we consider the distributed database example of Figure 5, we can find that skyline 3-sets corresponding to normal vectors $\theta_1 = (-1, 0)$ and $\theta_2 = (0, -1)$ are $P_1 = (4, 14)$ and $P_2 = (15, 5)$, respectively. We can see that both P_1 and P_2 are compromised 3-sets. So, Algorithm 3 modifies the values of P_1 and P_2 to $(6, 14)$ and $(15, 6)$, respectively. For preserving individual privacy, the coordinator sends the perturbed 3-sets $(6, 14)$ and $(15, 6)$ to the user instead of original 3-sets $(4, 14)$ and $(15, 5)$. The coordinator uses the original values of skyline s-sets for facet expansion. For example, the coordinator uses $(4, 14)$ and $(15, 5)$ for facet expansion instead of perturbed 3-sets $(6, 14)$ and $(15, 6)$.

4 EXPERIMENTS

We have implemented the proposed parallel computation of the skyline sets queries in a distributed database using Java Agent Development Framework. We have performed the experiment in a simulation environment of fifty databases created by ten PCs running on Windows OS and connected by an Ethernet switch. Each of the PCs has an Intel® Core2 Duo, 2 GHz CPU, and 3 GB main memory. We evaluate our proposed privacy preserving skyline sets queries algorithm in distributed environment on synthetic datasets. As benchmark databases, we use the databases proposed by Borzsonyi et al. [1], in which there are three types of synthetic data distributions: "correlated", "anticorrelated", and "independent". We consider data dimensionality between 2 to 5.

We first evaluate the effect of set size. Figure 13 shows the results of 2D, 3D, 4D, and 5D cases for datasets with 2 500 k data distributed among fifty databases. Databases are distributed among five clusters and each database contains around 50 k data. We observe that with the increases of s, query time also increases. This is because as s increases, the number of sets in convex skyline also increases.

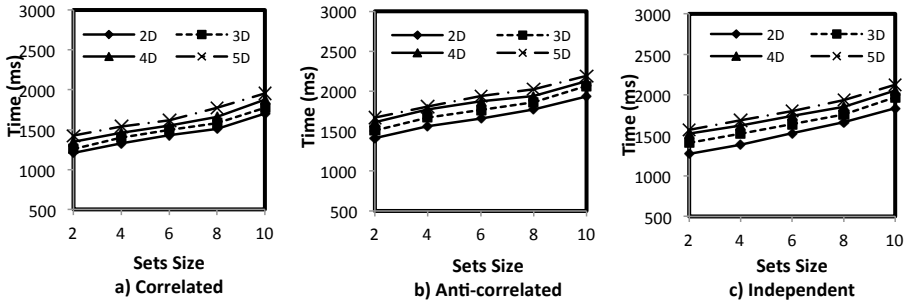


Figure 13. Time varying set size

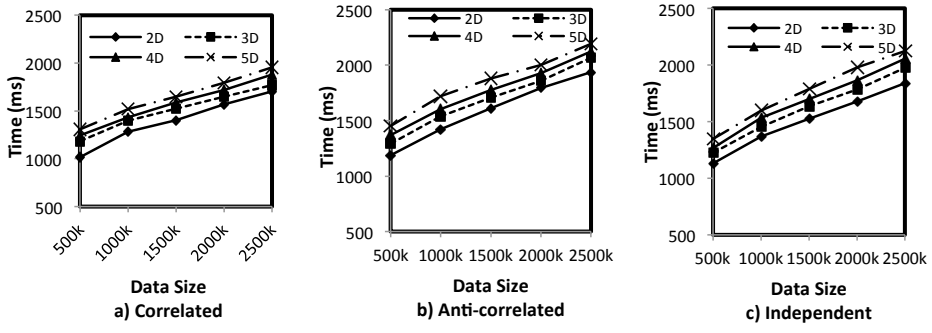


Figure 14. Time varying data size

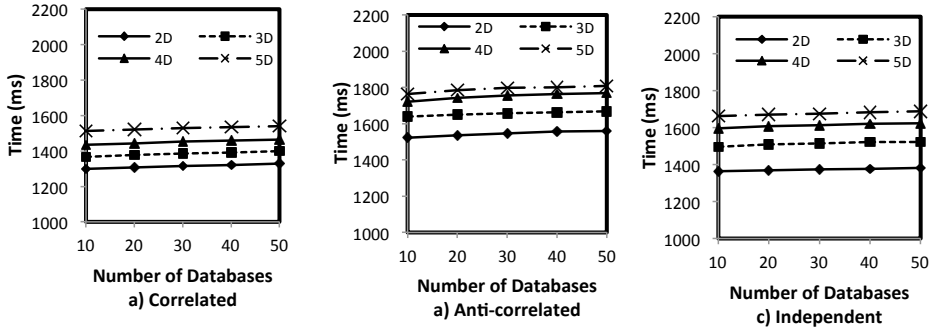


Figure 15. Time varying number of databases

In the next experiment, we evaluate the effect of data size. We used data with cardinality 500 k, 1 000 k, 1 500 k, 2 000 k, and 2 500 k. Same as in the previous experiment fifty databases are distributed among five clusters and each cluster contains ten databases. In case of 500 k, each database contains at least 10 k data. Similarly, for datasets 1 000 k, 1 500 k, 2 000 k, and 2 500 k each database has at least 20 k, 30 k, 40 k, and 50 k data, respectively. In this experiment, we set s to 10. Figure 14 shows the results. In this experiment, it is observed that response time increases with the increase of data set size. It is also observed that response time gradually increases if the dimension increases.

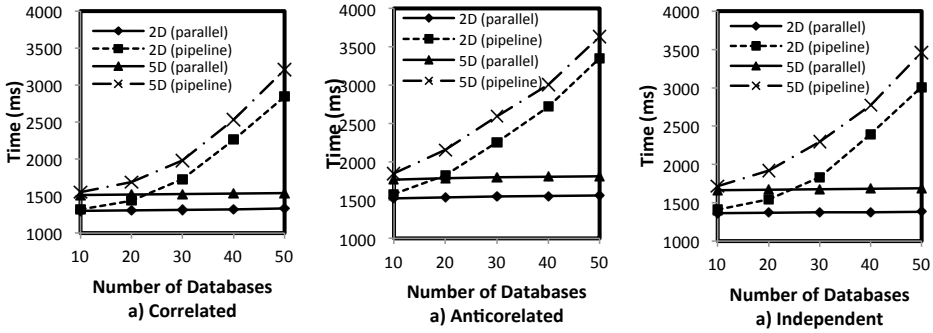


Figure 16. Comparison between parallel and pipeline computation

Next, we conduct the experiment to examine the effect of the number of DBs in the computation process. In this experiment, we distribute 2500k data to $m = 10$, 20, 30, 40, 50 databases. Here, we fix the number of clusters to five. For 10, 20, 30, 40, and 50 databases, each cluster contains two, four, six, eight, and ten databases, respectively. In this experiment, we set s to 4 and examine 2D, 3D, 4D, and 5D cases. Figure 15 shows the result. We find that the computation time is almost independent of the number of databases.

Finally, we conduct an experiment to examine the comparative performance of our method and pipeline computation method, a method where later agents need to wait for the completion of tasks by earlier agents which dramatically reduces the computation performance if there are many databases involved in the computation process. Same as in the previous experiment, we distribute 2 500 k data to $m = 10$, 20, 30, 40, 50 databases. In this experiment, we set s to 4 and examine 2D and 5D cases. From the result of Figure 16, it is found that when the number of databases is relatively small the computation time is almost the same in our method and pipeline computation method. However, as the number of databases increases, the pipeline execution method becomes slower while our proposed parallel computation method shows almost similar performance. The waiting delay of the later agents for the completion of tasks by the earlier agents is one reason of slowing down the computation. Another reason is that in pipeline execution approach many databases become idle due to the lack of available normal vectors.

5 RELATED WORKS

5.1 Skyline Query

Borzonyi et al. [1] first introduced the skyline operator into database systems and proposed Block Nested Loop (BNL), Divide-and-Conquer, and B-tree based algorithms. As a variant of BNL, Chomicki et al. [2] improved BNL algorithm with the help of a Sort-Filter-Skyline (SFS) algorithm. In SFS, data needs to be pre-sorted using a monotone scoring function, which can simplify the selection of skyline objects. Tan et al. [3] proposed two progressive algorithms: Bitmap and Index. The bitmap algorithm represents points in bit vectors and performs bit-wise operations. On the other hand, the index approach uses data transformation and B+-tree indexing. Kossmann et al. [4] proposed a Nearest Neighbor (NN) method. It selects skyline points by recursively invoking R*-tree based depth-first NN search over different data portions. Papadias et al. [5] proposed a Branch-and-Bound Skyline (BBS) method based on the best-first nearest neighbor algorithm. Godfrey et al. [6] provided a comprehensive analysis of previous skyline algorithms without indexing supports and proposed a new hybrid method with improvement.

5.2 Skyline Query in Distributed Environment

In [7], Wu et al. first address the problem of parallelizing skyline queries over a shared-nothing architecture. They provided two mechanisms: recursive region partitioning and dynamic region encoding for the execution of skyline queries. In their approach, a server starts the skyline computation on its data after receiving the results of other servers based on the partial order. It causes a waiting delay of the servers. In our work, such problem is localized within the cluster. Park et al. [8] introduced two parallel skyline algorithms in multicore architectures. The first one

is a parallel version of BBS algorithm. The second one is known as pskyline, which is based on skeletal parallel programming [24]. Gao et al. [9] proposed parallel computation of skyline queries in multi-disk environment using parallel R-trees. The core of their scheme is to visit more entries simultaneously and to enable effective pruning strategies. Cui et al. [10] introduced skyline queries in large-scale distributed environments without the assumption of any overlay structures and propose PaDSkyline algorithm. PaDSkyline is an algorithm that significantly reduces the response time by performing parallel processing over site groups produced by a partition algorithm. Within each group, it locally optimizes the query processing. It also improves the network transmission efficiency by performing early reduction of skyline candidates. Vlachou et al. [11] proposed an angle-based space partitioning scheme for parallel computation of skylines of data points using the hyperspherical coordinates of the data points. In their approach, data points are almost equally spread among the partitions which increases the average pruning power of data points.

Huang et al. [12] considered a setting with mobile devices communicating via an ad-hoc network (MANETs) and studied skyline queries that involved spatial constraints. In this approach, queries are forwarded through the whole MANET without routing information. They proposed a filtering based data reduction technique to reduce the data transfer among devices. However, in our work, we assume a wired large-scale distributed environment in which query results from each cluster are sent to the coordinator for computing skyline sets. In [13], Vlachou et al. studied the problem of subspace skyline processing in a super-peer network. In this approach, peers hold their data in an autonomous manner and collectively process skyline queries on subspaces. Hose et al. [14] introduced relaxed skylines in Peer Data Management Systems (PDMS). They proposed a strategy for processing relaxed skylines in distributed environments using distributed data summaries. For efficient computation of skyline, Wang et al. [15] use the z-curve method to map the multidimensional data to one dimensional values. The one-dimensional values are then assigned to peers connected in a tree overlay like BATON [25]. In this approach, the problem of load balancing arises. In particular, in their approach the peers near the origin of the axes need to process most of the queries. Li et al. [16] use a space partitioning method that is based on an underlying semantic overlay. Their approach shares the same drawbacks as [15]. Balke et al. [17] addressed skyline operation over web databases where different dimensions are stored in different data sites. Their algorithm first retrieves values in every dimension from remote data sites using sorted access in round-robin fashion on all dimensions. This continues until all dimension values of an object, called the terminating object, have been retrieved. Then, all non-skyline objects are filtered from all those objects with at least one dimension value retrieved. In [18], Fotiadou et al. proposed a bitmap approach for efficient subspace skyline computation in a distributed setting. The bitmap approach computes extended skylines that includes all points necessary for computing the skyline at any subspace. They presented an algorithm for computing extended skylines using a bitmap representation along with a storage efficient bucket-based variation of bitmap representation. Rocha et al. [19] introduced an ef-

ficient execution plan for distributed skyline query processing. In this paper, the authors proposed SkyPlan, a mechanism for querying servers consecutively. Their approach reduces the amount of transferred data and the number of queried servers.

Most research papers on parallel and distributed skyline query processing so far have the problem of load balancing. As a result few peers need to carry almost all the processing burden, while most other peers remain idle. Moreover, there is no consideration about individual privacy. In contrast, we introduce a parallelizing mechanism in which every server takes part in the computation simultaneously and preserves individual privacy.

5.3 Privacy Preserving Skyline Query

Although privacy of individual is an important issue in any computation, till now there is very little consideration about preserving individual privacy during skyline computation. As for the privacy issue, authors in [20] introduce Range to Ranges Skyline Query (R2R Skyline Query) and Point to Ranges Skyline Query (P2R Skyline Query) methods to deal with the privacy problems for Location-Based Services. This work is designed only for Location-Based Services and cannot be applied for general numerical databases.

Su et al. [21] considered top-k combinatorial skyline queries. Their top-k combinatorial skyline problem is to compute the skyline of all s-sets ($s = 1, \ldots, k$). Our skyline s-sets are not a subset of their top-k combinatorial skyline. Their results can preserve privacy in a sense if they eliminate combinatorial skyline objects with small cardinality. However, their efficient algorithm is not suitable for privacy-aware distributed databases since it is an incremental algorithm and requires individual record values to prune unnecessary search.

Our previous works [22, 23] can compute skyline sets. In [22], we considered skyline sets computation from a single database. Our work in [23] considered a distributed database where computation was carried out in pipeline fashion. Here, the computation process becomes comparatively slower if there are large number of servers in skyline computation. Both of our previous works also have the limitation that there is no protection mechanism against compromisable situations.

Our work in this paper takes enough protection mechanism against compromisable situations. Moreover, due to proper parallelism, the computation time of the propose algorithm in this paper is almost independent of the number of servers involved in skyline sets computation.

6 CONCLUSIONS

In privacy aware environment in which we are only allowed to disclose aggregated values of objects, skyline sets queries can be a promising alternative for analyzing and making important decisions. With the rapid growth of network infrastructure, distributed databases are becoming popular. In privacy aware environment, each

owner of distributed databases does not want to disclose the attribute values of her/his databases to others. Therefore, we proposed an agent-based algorithm for computing skyline sets queries in a parallel manner from distributed databases in this paper.

The proposed algorithm can efficiently calculate skyline sets from the distributed databases. Experimental results demonstrate that the proposed algorithm for skyline sets queries is scalable enough to handle large and high-dimensional datasets. The performance of our proposed approach is almost independent of the number of databases involved in skyline sets queries. We have also proposed a privacy protection mechanism in which we detect compromisable sets and perturb such sets so that individual records values cannot be identified.

In this work, we assume that all the attributes of the databases are numerical. In future, we hope to develop parallel algorithms for skyline sets queries from databases with categorical attributes and from spatio-temporal databases.

Acknowledgement

This work was supported by KAKENHI (19500123). Mohammad Shamsul Arefin was supported by the scholarship of MEXT Japan.

REFERENCES

[1] Borzonyi, S.—Kossmann, D.—Stocker, K.: The Skyline Operator. Proceedings of the 17th International Conference on Data Engineering, Heidelberg, Germany, April 2–6, 2001, pp. 421–430.

[2] Chomicki, J.—Godfrey, P.—Gryz, J.—Liang, D.: Skyline with Presorting. Proceedings of the 19th International Conference on Data Engineering, Bangalore, India, March 5–8, 2003, pp. 717–816.

[3] Tan, K. L.—Eng, P. K.—Ooi, B. C.: Efficient Progressive Skyline Computation. Proceedings of 27th International Conference on Very Large Data Bases, Roma, Italy, September 11–14, 2001, pp. 301–310.

[4] Kossmann, D.–Ramsak, F.—Rost, S.: Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. Proceedings of 28th International Conference on Very Large Data Bases, Hong Kong, China, August 20–23, 2002, pp. 275–286.

[5] Papadias, D.—Tao, Y.—Fu, G.—Seeger, B.: An Optimal and Progressive Algorithm for Skyline Queries. Proceedings of ACM SIGMOD Conference, San Diego, California, June 9–12, 2003, pp. 467–478.

[6] Godfrey, P.—Shipley, R.— Gryz, J.: Maximal Vector Computation in Large Data Sets. Proceedings of 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30–September 2, 2005, pp. 229–240.

[7] Wu, P.—Zhang, C.—Feng, Y.—Zhao, B. Y.—Agrawal, D.—Abbadi, A. E.: Parallelizing Skyline Queries for Scalable Distribution. Proceedings of 10th In-

ternational Conference on Extending Database Technology, Munich, Germany, March 26–31, 2006, pp. 112–130.

[8] Park, S.—Kim, T.—Park, J.—Kim, J.—Im, H.: Parallel Skyline Computation on Multicore Architectures. Proceedings of the 25th International Conference on Data Engineering, Shanghai, China, March 29–April 2, 2009, pp. 760–771.

[9] Gao, Y.—Chen, G.—Chen, L.—Chen, C.: Parallelizing Progressive Computation for Skyline Queries in Multi-Disk Environment. Proceedings of International Conference of Database and Expert Systems Applications (DEXA), Krakow, Poland, September 4–8, 2006, pp. 697–706.

[10] Cui, B.—Lu, H.,—Xu, Q.—Chen, L.—Dai, Y.—Zhou, Y.: Parallel Distributed Processing of Constrained Skyline Queries by Filtering. Proceedings of the 24th International Conference on Data Engineering, Cancun, Mexico, April 7–12, 2008, pp. 546–555.

[11] Vlachou, A.—Doulkeridis, C.—Kotidis, Y.: Angle-Based Space Partitioning for Efficient Parallel Skyline Computation. Proceedings of ACM SIGMOD Conference, Vancouver, Canada, June 9–12, 2008, pp. 227–238.

[12] Huang, Z.—Jensen, C. S.—Lu, H.—Ooi, B. C.: Skyline Queries Against Mobile Lightweight Devices in MANETs. Proceedings of the 22nd International Conference on Data Engineering, Atlanta, GA, USA, April 3–7, 2006, pp. 66, 2006.

[13] Vlachou, A.—Doulkeridis, C.—Kotidis, Y.—Vazirgiannis, M.: SKYPEER: Efficient Subspace Skyline Computation over Distributed Data. Proceedings of the 23rd International Conference on Data Engineering, Istanbul, Turkey, April 15–20, 2007, pp. 416–425.

[14] Hose, K.—Lemke, C.—Sattler, K. U.: Processing Relaxed Skylines in PDMS Using Distributed Data Summaries. Proceedings of International Conference on Information and Knowledge Management, Arlington, Virginia, USA, November 6–11, 2006, pp. 425–434.

[15] Wang, S.,—Ooi, B. C.—Tung, A. K.—Xu, L.: Efficient Skyline Query Processing on Peer-to-Peer Networks. Proceedings of the 23rd International Conference on Data Engineering, Istanbul, Turkey, April 15–20, 2007, pp. 1126–1135.

[16] Li, H.—Tan, Q.—Lee, W. C.: Efficient Progressive Processing of Skyline Queries in Peer-to-Peer Systems. Proceedings of 1st International Conference on Scale Information Systems, New York, NY, USA, 2006, pp. 26.

[17] Balke, W. T.—Guntzer, U.—Zheng, J. X.: Efficient Distributed Skylining for Web Information Systems. Proceedings of 9th International Conference on Extending Database Technology, Heraklion, Crete, Greece, March 14–18, 2004, pp. 256–273.

[18] Fotiadou, K.—Pitoura, E.: BITPEER: Continuous Subspace Skyline Computation with Distributed Bitmap Indexes. Proceedings of International Workshop on Data Management in Peer-to-Peer Systems, Nantes, France, March 25–30, 2008, pp. 35–42.

[19] Rocha, J. B.—Vlachou, A.—Doulkeridis, C.—Norvag, K. I.: Efficient Execution Plans for Distributed Skyline Query Processing. Proceedings of 14th International Conference on Extending Database Technology, Uppsala, Sweden, March 21–24, 2011, pp. 271–282.

[20] Qiao, Z.—Gu, J.—Lin, X.—Chen, J.: Privacy-Preserving Skyline Queries in LBS. Proceedings of International Conference on Machine Vision and Human-Machine Interface, Kaifeng, China, April 24–25, 2010, pp. 499–504.

[21] Su, I. F.—Chung, Y. C.—Lee, C.: Top-k Combinatorial Skyline Queries. Proceedings of 15$^{\text{th}}$ Database Systems for Advanced Applications (DASFAA), Tsukuba, Japan, April 1–4, 2010, pp. 79–93.

[22] Siddique, M. A.—Morimoto, Y.: Algorithm for Computing Convex Skyline Objectsets on Numerical Databases. IEICE Transaction on Information and Systems, Vol. E93-D, 2010, No. 10, pp. 2709–2716.

[23] Morimoto, Y.—Arefin, M. S.—Siddique, M. A.: Agent-Based Anonymous Skyline Set Computation in Cloud Databases. International Journal of Computational Science and Engineering, Vol. 7, 2012, No. 1, pp. 73–81.

[24] Rabhi, F. A.—Gorlatch, S.: Patterns and Skeletons for Parallel and Distributed Computing. Springer-Verlag, 2003.

[25] Jagadish, H. V.—Ooi, B. C.—Vu, Q. H.: BATON: A Balanced Tree Structure for Peer-to-Peer Networks. Proceedings of 31$^{\text{st}}$ International Conference on Very Large Data Bases, Trondheim, Norway, August 30–September 2, 2005, pp. 661–672.

[26] Morimoto, Y.—Fukuda, T.—Matsuzawa, H.—Yoda, K.—Tokuyama, T.: Algorithms for Mining Association Rules for Binary Segmentations of Huge Categorical Databases. Proceedings of 24$^{\text{th}}$ International Conference on Very Large Data Bases, New York City, USA, August 24–27, 1998, pp. 380–391.

Mohamad Shamsul Arefin received his B. Sc. in Engineering in Computer Science and Engineering from Khulna University, Khulna, Bangladesh in 2002, and his M. Sc. in Engineering in Computer Science and Engineering in 2008 from Bangladesh University of Engineering and Technology (BUET), Bangladesh. He received his Doctor of Engineering Degree from Hiroshima University with support of the scholarship of MEXT, Japan in 2013. He is a member of Institution of Engineers, Bangladesh (IEB) and is currently working as an Associate Professor in the Department of Computer Science and Engineering, Chittagong University of Engineering and Technology, Chittagong, Bangladesh. His research interest includes privacy preserving data mining, cloud privacy, multilingual data management, semantic web, and object oriented system development.

Yasuhiko Morimoto is an Associate Professor at Hiroshima University. He received B. E., M. E., and Ph. D. from Hiroshima University in 1989, 1991, and 2002, respectively. From 1991 to 2002, he had been with IBM Tokyo Research Laboratory where he worked for data mining project and multimedia database project. Since 2002, he has been with Hiroshima University. His current research interests include data mining, machine learning, geographic information system, and privacy preserving information retrieval.