Agent-Based Convex Skyline Set Query for Cloud Computing Environment

[Extended Abstract]

Yasuhiko Morimoto
Hiroshima University, Japan
morimoto@mis.hiroshima-u.ac.jp

MD. Anisuzzaman Siddique
University of Rajshahi, Bangladesh
anis_cst@yahoo.com

MD. Shamsul Arefin*
Hiroshima University, Japan
d105660@hiroshima-u.ac.jp

ABSTRACT

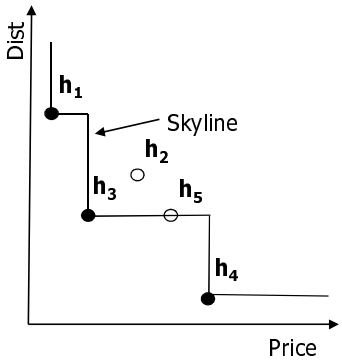
Given a set of objects, a skyline query finds the objects that are not dominated by another object. A skyline query helps us to filter unnecessary information efficiently and gives us clues for various decision making tasks. On the other hand, we have to be aware of individual's privacy. In privacy aware environments, we usually have to hide individual record's values even though there is no ID information in the table. In such situation, we cannot use conventional skyline queries. To handle the privacy problem, we considered a skyline query for sets of objects in a database. Let s be the number of objects in each set and n be the number of objects in the database. There are ${}_nC_s$ sets in the database. We consider an efficient algorithm for computing convex skyline of the ${}_nC_s$ sets, which we call "convex skyline sets". We further expand the idea to use the skyline set query in a cloud computing environment in this paper. We propose a method for computing a skyline set query from distributed databases without disclosing individual records to others. There is no doubt that most of the cloud service providers do not want to disclose any individual record in their database. The proposed method utilize an agent computing framework and solves the privacy problems of skyline queries in cloud computing environments.

1. INTRODUCTION

Skyline queries retrieve a set of skyline objects so that a user can choose promising objects or eliminate unnecessary objects. Given a k-dimensional database DB, an object p is said to be in skyline of DB if there is no object q in DB such that q is better than p in all k dimensions. If there exists such an object q, then we say that p is dominated by q or q dominates p. Figure 1 shows a typical example of skyline objects. The table in Figure 1 is a list of hotels, each of which contains two numerical attributes "distance" and "price". In the list, the best choice usually comes from the skyline, i.e., one of $\{h_1, h_3, h_4\}$ (See Figure 1 (b)). A number of efficient algorithms for computing skyline have been reported in the literature [1, 8, 2, 5, 6, 3].

ID	Price	Dist
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

Recently, the individuals' privacy preserving is one of the important data management issues. In many tables in a database, we have to hide individual record's values to preserve privacy even though there is no ID information in the table. In such situation, we cannot use conventional skyline queries.

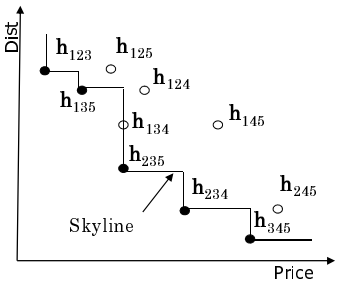
Therefore in [7], we considered a skyline query for sets of objects in a database. Let s be the number of objects in each set and n be the number of objects in the database. There are ${}_nC_s$ sets in the database. We consider an efficient algorithm for computing convex skyline of the ${}_nC_s$ sets, which we call "convex skyline sets". This function does not disclose individual values of an object. Instead, it discloses aggregated values of s objects. It will be one of the most promising alternatives for decision making in a privacy aware environment.

Figure 2 (a) is a list of 3-sets, in which all of the combinations of three hotels are listed. The "h_{123}" denotes a set of $\{h_1,h_2,h_3\}$. "Distance" and "Price" of "h_{123}" are the sum

*On leave from Chittagong University of Engineering and Technology, Bangladesh.

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

of the "Distance" and "Price" of respective hotels in the set. The skyline of the combinations of three hotels are $\{h_{123}, h_{135}, h_{235}, h_{234}, h_{345}\}$. If one wants to know the cheapest hotel, she/he can find that the cheapest set is h_{123} from the skyline and can easily imagine that the price of the cheapest hotel is around 4, since price of the cheapest 3-set h_{123} is 12. Similarly, if one prefers cheaper and closer, she/he may choose h_{235} from the skyline and can easily imagine the value of the preferable choice from the aggregated values.

In this paper, we expanded the skyline set queries to distributed databases. Recent development of network infrastructure makes it possible to provide various services over the Internet, which we call "cloud computing." In the cloud computing environment, each service provider is located in different sites and collects information in her/his own database. Some of the services collect information in a distributed database. Hence, users' information are separately stored in the Internet. If we can compute skyline set queries over the cloud, we can find valuable knowledge.

However, a database owner, in general, does not want to disclose any record's value of her/his database to other database owners. This is the most significant problem for users of skyline queries. Therefore, we consider an agent based method that computes a skyline set query without disclosing any individual record to others. The proposed method solves the privacy problems of skyline queries in cloud computing environment.

2. SKYLINE SETS PROBLEM

We consider the database DB having k attributes and n records. Let a_1, a_2, ..., a_k be the k attributes of DB. Without loss of generality, we assume that smaller value in each attribute is better.

Let s-set be an object set whose size is s. We assume s is a relatively small number such that $2 \leq s \leq 10$ though we can compute s-sets for much larger s. Let S be a database of all s-sets in DB. Note that the number of record in S, i.e., the number of s-sets in DB, is ${}_nC_s = \frac{n!}{(n-s)!s!}$, we denote the number by $|S|$. We assume a virtual database of S on the k dimensional space of DB. Each record of the database is an s-set whose value of each attribute (dimension) is the sum of s values of corresponding s objects. We denote $p.a_l$ as the l-th attribute value of a record p in S.

A s-set $p \in S$ is said to dominate another s-set $q \in S$, denoted as $p \leq q$, if $p.a_r \leq q.a_r$ $(1 \leq r \leq k)$ for all k attributes and $p.a_t < q.a_t$ $(1 \leq t \leq k)$ for at least one attribute. We call such p as dominant s-set and q as dominated s-set between p and q.

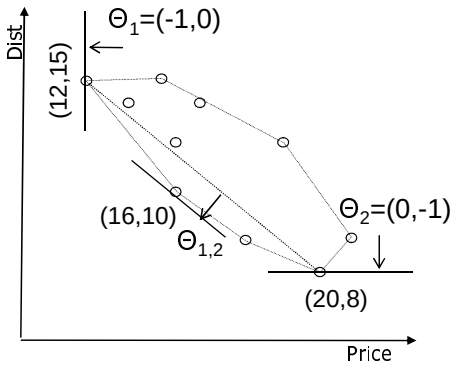


Figure 3: Touching Oracle in 2D Space

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.

2.1 Convex Skyline

We can consider a record in S to be a point in k-dimensional vector space. Convex hull for the set of $|S|$ points is the minimum polyhedron containing the set.

Consider the examples of Figure 1 and Figure 2 again. There are 10 points in S if $s = 3$. Since there are two attributes in the database, those 10 points are in two-dimensional space as in Figure 2 (b). The dotted polygon in Figure 3 is the convex hull of the 10 points.

In Figure 3, $(\underline{12}, 15)$ and $(20, \underline{8})$ are the point that has the minimum value of attribute "Price" and "Distance", respectively. We call such points that have the minimum value of an attribute as initial points. Notice that such points must be in the convex hull. We call the line (hyperplane) between the two initial points (k initial points) the initial facet.

Among all points in the convex hull, points that lie outside of the initial facet are skyline objects and we call such points convex skyline objects. In k-dimensional space, we compute such initial hyperplane surrounded by k points as the initial facet. Then, we compute convex skyline objects that lie in the convex hull 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 that lies in both the convex hull and the skyline of S.

3. ALGORITHM FOR COMPUTING CONVEX SKYLINE SETS

In this section, we assume that there is a single database that is the union of all distributed databases and explain how to compute the convex skyline sets from the union.

If we compute all of the s-sets from the original database and make a database containing $|S|$ records, the problem can be solved by a conventional skyline query algorithm. However,

Table 1: Inner Product with Tangent Lines

$\mathbf{o}$	$(\mathbf{\Theta_1}, \mathbf{o})$	$(\mathbf{\Theta_2}, \mathbf{o})$	$(\mathbf{\Theta_{1,2}}, \mathbf{o})$
$\mathbf{h_1}$	-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

$|S|$ is unacceptably large when the original database size is large. Therefore, we consider an algorithm for finding convex skyline sets without computing $|S|$ s-sets.

3.1 Touching Oracle

Each s-set in S can be represented as a k-dimensional point $\mathbf{x} = (x_1, x_2, ..., x_k)$ where x_i $(1 \leq i \leq k)$ is the sum of the i-th attribute's value of the s objects in DB.

Touching oracle function is a method to compute a point on the convex hull without generating S. It computes the tangent point of the convex hull of S and a $(k-1)$-dimensional hyperplane directly from DB.

In the hotel example, there are five records in the original databases DB as in Figure 1 (a). Each of the five records is represented as a two-dimensional point, which we call an atomic point, which is denoted as $\mathbf{o}$.

Assume there is a $(k-1)$-dimensional hyperplane (which is a line if $k = 2$), whose normal vector is $\Theta_1 = (-1, 0)$ in the two-dimensional space. In order to find the tangent point with the 1-dimensional hyperplane (line) and the convex hull without precomputing all points in S, we compute $(\Theta_1, \mathbf{o})$, i.e., inner products of the normal vector and each atomic point as in the second column of Table 1. We choose the top three inner products, i.e., $\{h_1, h_2, h_3\}$. Those top three inner products composes the tangent point $(12, 15)$, which is the 3-set, h_{123}. Similarly, for a line with $\Theta_2 = (0, -1)$, we can find $\{h_3, h_4, h_5\}$ is the top three in the inner products of $(\Theta_2, \mathbf{o})$. It composes the tangent point $(20, 8)$, which is the tangent point of the convex hull and the 1-dimensional hyperplane (line) whose normal vector is Θ_2.

Like this way, 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 top s inner products from n atomic points in DB. Since s is negligible small constant, we can compute the tangent point by scanning n atomic points only once, which is $O(n)$.

3.2 Convex Hull Search

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

Next, we compute the normal vector of the initial facet. In the example above, we have initial two tangent points: we have $p_1 = (12, 15)$ with the normal vector $\Theta_1 = (-1, 0)$ and we have $p_2 = (20, 8)$ with the normal vector $\Theta_2 = (0, -1)$. 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 recursively compute tangent points for each of the expanded facet. If we find 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 convex skyline s-sets. We can apply this recursive operation for higher k-dimensional space [4]. In the k-dimensional case, new tangent point, which is found by the touching oracle, divides the initial facet into k facets. In high dimensional case, the normal vector of each facet can be computed as follows:

Three Dimensional Case:

Assume we have a facet surrounded by following three points:

$$P1 = (p1_1, p1_2, p1_3)$$

$$P2 = (p2_1, p2_2, p2_3)$$

$$P3 = (p3_1, p3_2, p3_3)$$

We assume that P1, P2 and P3 are clockwise order when we look the facet from outside of the convex hull. Now, we can compute two edge vectors by using the three points as follows. Suppose the edges are following vectors.

$$V1 = (v1_1, v1_2, v1_3) = (p2_1, p2_2, p2_3) - (p1_1, p1_2, p1_3)$$

$$V2 = (v2_1, v2_2, v2_3) = (p3_1, p3_2, p3_3) - (p1_1, p1_2, p1_3)$$

The outside normal vector of this facet is computed as the expansion of the following symbolic determinant.

$$V1 \otimes V2 = \begin{vmatrix} e_1 & e_2 & e_3 \\ v1_1 & v1_2 & v1_3 \\ v2_1 & v2_2 & v2_3 \end{vmatrix}$$

In the formula, e_1, e_2, and e_3 are the elementary vectors (1,0,0), (0,1,0) and (0,0,1) respectively. Using this normal vector, we can divide this facet into three facets if we can find a new tangent point outside of the facet by the touching oracle function. If P is found outside of the facet, then the three new facets are as follows:

$$(P1, P, P3)$$

$$(P1, P2, P)$$

$$(P, P2, P3)$$

The normal vectors of these three facets are

$$(P - P1) \otimes (P3 - P1)$$

$$(P2 - P1) \otimes (P - P1)$$

$$(P2 - P) \otimes (P3 - P)$$

if points in each facet are clockwise order when we look the facet from outside of convex hull.

Four Dimensional Case:

We can use the idea into higher dimensional case analogically.

Assume that we have a facet surrounded by four points as follows:

$$P1 = (p1_1, p1_2, p1_3, p1_4)$$

$$P2 = (p2_1, p2_2, p2_3, p2_4)$$

$$P3 = (p3_1, p3_2, p3_3, p3_4)$$

$$P4 = (p4_1, p4_2, p4_3, p4_4)$$

Using similar operations of 3D case, we can compute three vectors as follows:

$$V1 = (v1_1, v1_2, v1_3, v1_4) = P2 - P1$$

$$V2 = (v2_1, v2_2, v2_3, v2_4) = P3 - P1$$

$$V3 = (v3_1, v3_2, v3_3, v3_4) = P4 - P1$$

Then, the normal vector that directs outside can be computed as the expansion of the following determinant.

$$V1 \otimes V2 \otimes V3 = \begin{vmatrix} e_1 & e_2 & e_3 & e_4 \\ v1_1 & v1_2 & v1_3 & v1_4 \\ v2_1 & v2_2 & v2_3 & v2_4 \\ v3_1 & v3_2 & v3_3 & v3_4 \end{vmatrix}$$

In the determinant, the value of e_1, e_2, e_3 and e_4 are (1,0,0,0), (0,1,0,0), (0,0,1,0), and (0,0,0,1), respectively. If P is found outside of the facet, then the four new facets are as follows:

$$(P1, P2, P3, P)$$

$$(P1, P2, P, P4)$$

$$(P1, P, P3, P4)$$

$$(P, P2, P3, P4)$$

The normal vectors of these four facets are as follows:

$$(P2 - P1) \otimes (P3 - P1) \otimes (P - P1)$$

$$(P2 - P1) \otimes (P - P1) \otimes (P4 - P1)$$

$$(P - P1) \otimes (P3 - P1) \otimes (P4 - P1)$$

$$(P2 - P) \otimes (P3 - P) \otimes (P4 - P)$$

k-Dimensional Case:

We can expand above operations for k-dimensional case. Assume we have a facet surrounded by following k points.

$$P1 = (p1_1, p1_2, ..., p1_k)$$

$$P2 = (p2_1, p2_2, ..., p2_k)$$

$$\cdots$$

$$Pk = (pk_1, pk_2, ..., pk_k)$$

We can calculate $(k-1)$ vectors like $V1$, $V2$, $\cdots$, $V(k-1)$. Then, the normal vector of the facet that directs outside can be computed as the expansion of the following determinant.

$$V1 \otimes \cdots \otimes V(k-1) = \begin{vmatrix} e_1 & \cdots & e_k \\ v1_1 & \cdots & v1_k \\ \cdots & \cdots & \cdots \\ v(k-1)_1 & \cdots & v(k-1)_k \end{vmatrix}$$

If P is found outside of the facet, then the k new facets are as follows:

$$(P, P2, \cdots, Pk-1, Pk)$$

$$(P1, P, \cdots, Pk-1, Pk)$$

$$\cdots$$

$$(P1, P2, \cdots, Pk-1, P)$$

The normal vectors of these k facets are as follows:

$$((P2 - P) \otimes \cdots \otimes (Pk - 1 - P) \otimes (Pk - P))$$

$$((P - P1) \otimes \cdots \otimes (Pk - 1 - P1) \otimes (Pk - P1))$$

$$\cdots$$

$$((P2 - P1) \otimes \cdots \otimes (Pk - 1 - P1) \otimes (P - P1))$$

4. PRIVACY PRESERVING SKYLINE SETS QUERY IN DISTRIBUTED DATABASES

In a cloud computing environment, databases are distributed in the network. If we can use skyline queries across the distributed databases, we can include more information than a single database and hence we have more chances to find unknown knowledge. However, since we have to be aware of privacy, each database owner does not want to disclose a record in her/his database, even though she/he can understand the importance of knowledge from skyline queries.

As we have mentioned above, skyline set queries do not disclose a record value of a database. In this section, we present an agent-based method for computing skyline set queries from distributed databases, in which any individual record values are not disclosed to others.

4.1 Problem Formulation

We assume there are m databases, which share the same schema, in a network. Let DB_1, DB_2, $\cdots$, DB_m be the databases. Each database has a view table whose schema has following columns: ID, a_1, a_2, $\cdots$, a_k, where ID is the

ID	a_1	a_2
$\mathbf{o_1}$	6	3
$\mathbf{o_2}$	3	5
$\mathbf{o_3}$	7	5
$\mathbf{o_4}$	5	8
$\mathbf{o_5}$	4	6

DB_1

ID	a_1	a_2
$\mathbf{o_6}$	7	8
$\mathbf{o_7}$	8	3
$\mathbf{o_8}$	4	9
$\mathbf{o_9}$	3	7
$\mathbf{o_{10}}$	8	5

DB_2

ID	a_1	a_2
$\mathbf{o_{11}}$	5	5
$\mathbf{o_{12}}$	2	6
$\mathbf{o_{13}}$	6	4
$\mathbf{o_{14}}$	9	7
$\mathbf{o_{15}}$	6	8

DB_3

Figure 4: Distributed Database Example

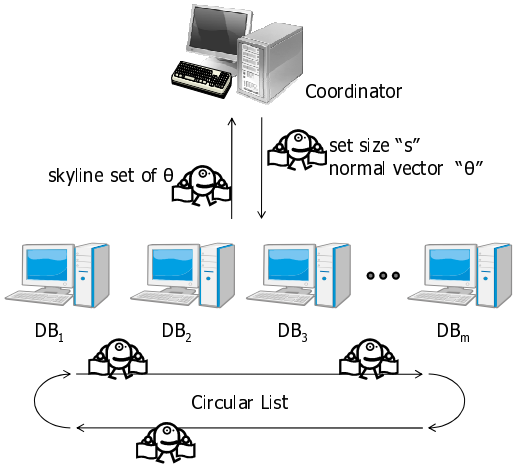


Figure 5: Agent-Based Computation

primary key attribute and a_i $(i = 1, ..., k)$ are k-dimensional numerical attributes. The problem is to compute skyline sets from the union of those m databases without disclosing individual record values to others.

Figure 4 is an example of a distributed database consists of three local databases, DB_1, DB_2, and DB_3. Each local database contains five two-dimensional records.

In the example, 3-set that corresponds to normal vector $(-1, 0)$ is $\{\mathbf{o_2}, \mathbf{o_9}, \mathbf{o_{12}}\}$, whose coordinate values in the two-dimensional space is $(8, 18)$. Similarly, 3-set that corresponds to normal vector $(0, -1)$ is $\{\mathbf{o_1}, \mathbf{o_7}, \mathbf{o_{13}}\} = (20, 10)$.

Given s, we have to compute all such convex skyline s-sets from a distributed database.

4.2 Agent-Based Computation

We assume there is a coordinator who is responsible for performing the convex hull search, which is mentioned in Section 3.2. The coordinator creates agents and provides a normal vector to each agent. Each agent computes the touching oracle function, which is mentioned in Section 3.1. In order to carry out the touching oracle function, each agent travels each of the local databases along a pre-defined circular list. Figure 5 shows the overview of the agent-based computation.

Assume that we have to compute convex skyline 3-set query from the distributed databases of Figure 4. The coordinator generates two agents for the two initial normal vectors, i.e.,

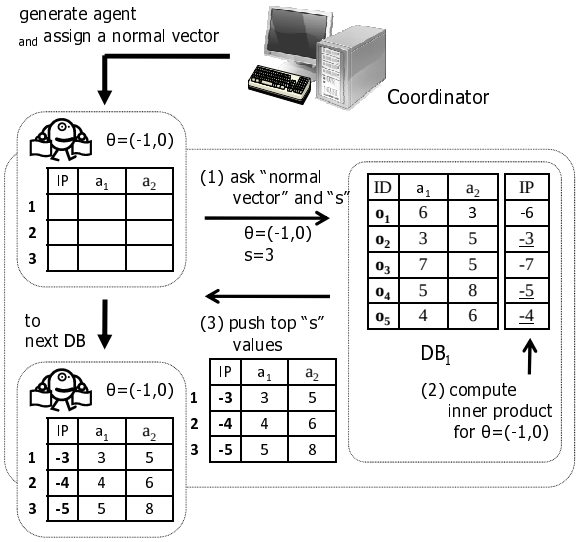


Figure 6: Agent with $(-1, 0)$ at DB_1

$(-1, 0)$ and $(0, -1)$.

Each agent has an 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. One of the initial two agents has $(-1, 0)$. The other agent has $(0, -1)$. Each of the two agents starts traveling the circular list to compute the touching oracle result of the assigned normal vector.

Figure 6 shows the touching oracle computation of the agent having $(-1, 0)$ at DB_1. When an agent arrives to a server of a distributed database, the server (1) asks the normal vector of the agent. Next, the server (2) computes inner product for each record of the database. The server, then, (3) pushes the top 3 inner product values to the agent. In the example of Figure 6, $(-\mathbf{3}, 3, 5)$, $(-\mathbf{4}, 4, 6)$, and $(-\mathbf{5}, 5, 8)$ are pushed to the priority queue of the agent. In this example, all the pushed values are stored in the top-3 priority queue. After these three procedures, the agent goes to the next server.

Note that the server cannot see the content of the priority queue of the agent during these three procedures.

Figure 7 shows the touching oracle computation of the agent at DB_2. In this server, $(-\mathbf{3}, 3, 7)$, $(-\mathbf{4}, 4, 9)$, and $(-\mathbf{7}, 7, 8)$ are pushed. In each of the push operations, the agent compares the least one to new one and replaces the priority queue (pop the least one and push the new one) if necessary. After the push operations of this server, the agent goes to the next server with the top 3 values so far.

Figure 8 shows the touching oracle computation of the agent at DB_3. After the necessary procedures of this server, the agent returns to the coordinator. When an agent returns, the coordinator asks the normal vector and the corresponding skyline 3-set. In this example, the coordinator receives $\Theta_1 = (-1, 0)$ and $p_1 = (8, 18)$ from the agent.

Similarly, when the agent with $(0, -1)$ returns from the

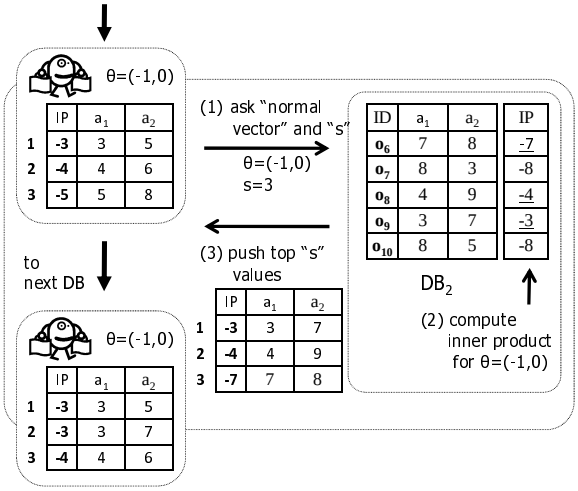


Figure 7: Agent with $(-1, 0)$ at DB_2

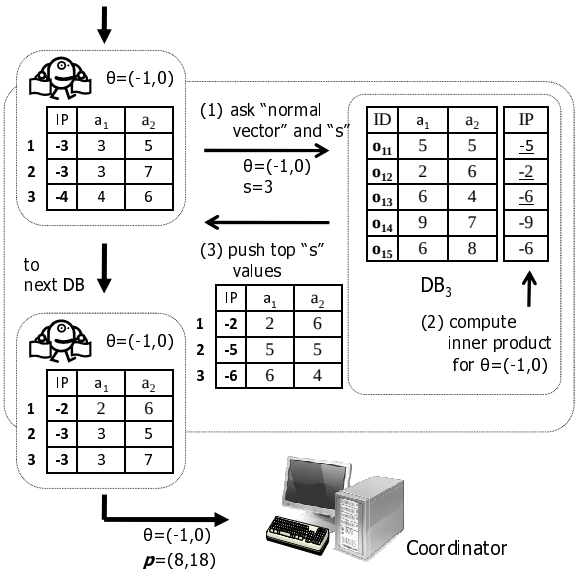


Figure 8: Agent with $(-1, 0)$ at DB_3

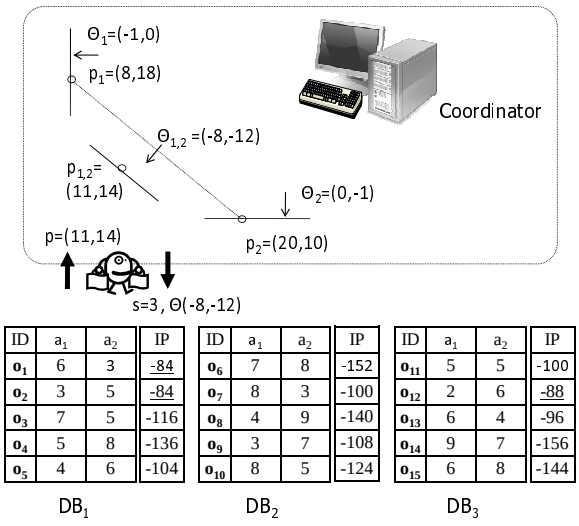


Figure 9: Agent-Based Convex Hull Search

travel, the coordinator receives $\Theta_2 = (0, -1)$ and $p_2 = (20, 11)$.

Note that the coordinator cannot see the content of the priority queue of each agent.

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, $p_1 = (8, 18)$, which is found by $\Theta_1 = (-1, 0)$, and $p_2 = (20, 10)$, which is found by $\Theta_2 = (0, -1)$, are two surrounding points of the initial facet (line segment between p_1 and p_2). The coordinator computes the normal vector $\Theta_{1,2} = (-8, -12) = (-(18 - 19), 8 - 20)$ from $p_1 = (8, 18)$ and $p_2 = (20, 10)$. Then, the coordinator generates another agent that has the normal vector $\Theta_{1,2} = (-8, -12)$.

The agent travels the distributed databases and comes back with skyline 3-set, which is $p_{1,2} = (11, 14)$, which is composed of $\{o_1, o_2, o_{12}\}$ as in Figure 9. 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. The coordinator recursively computes a normal vector for each facet and generates agents. The coordinator continues the convex hull search of Section 3.2 by using agents.

5. IMPLEMENTATION AND EXPERIMENT

We implemented the proposed skyline set queries function in a distributed database. We used Java Agent Development Framework (JADE) running on eight Windows PCs which are connected by an Ethernet switch. Each of the PCs has an Intel(R) Core2 Duo, 2 GHz CPU, and 3 GB main memory. All database servers including the coordinator who participate the skyline set queries join in the same agent platform. In this framework, each server creates a container of the agent platform, in which there are agents. Agents circulate the information, which we call "agent" in Section 4 that has a normal vector and a priority queue, along the circular list.

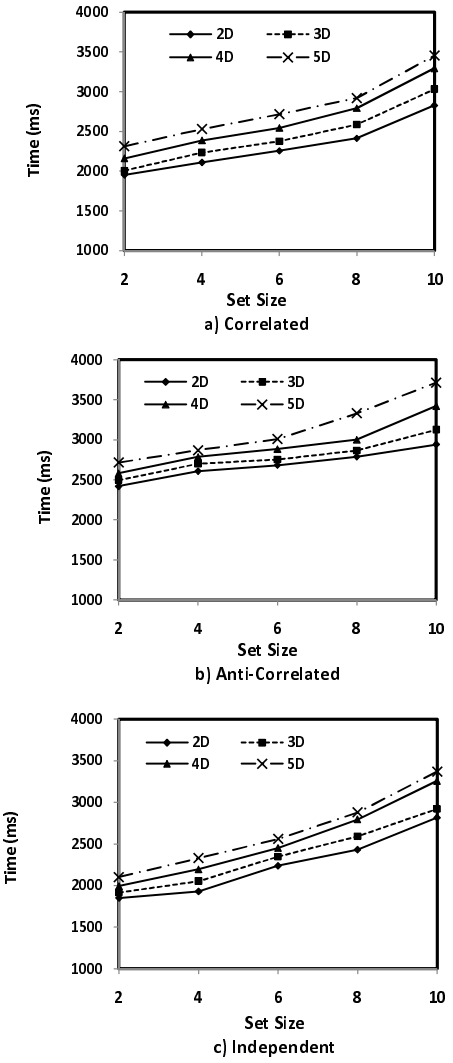


Figure 10: Time Varying Set Size

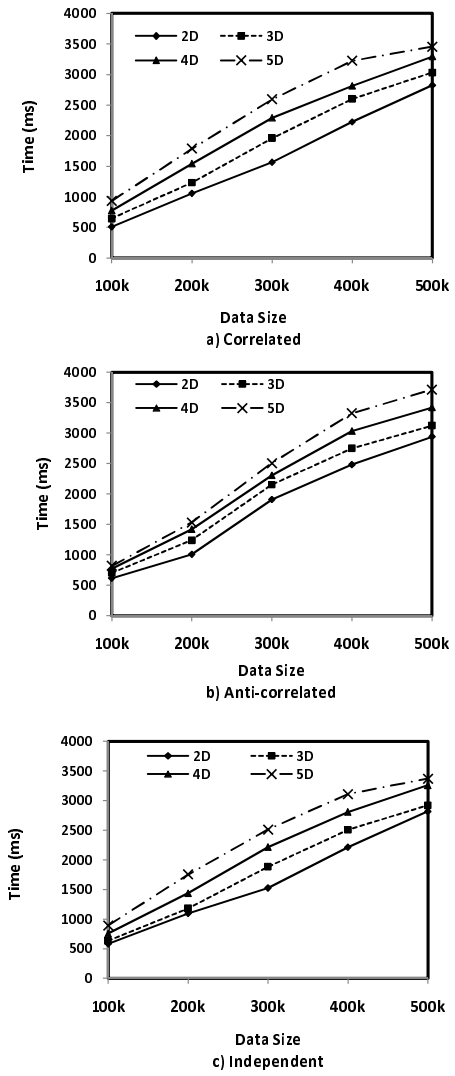


Figure 11: Time Varying Data Size

We conduct a series of experiments to evaluate the performance of the implemented method using different types of 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", "anti-correlated", and "independent".

We first evaluate the response time. Figure 10 shows the results of 2D, 3D, 4D and 5D cases for datasets with 500k data distributed to eight servers so that each server contains at least 50k data. We observe that proposed method becomes slower if "s" increases. If "s" increases, the number of sets in convex skyline also increases, which seems to be the main cause of the result.

Next, we evaluate the effect of dataset size. We used datasets with cardinality 100k, 200k, 300k, 400k and 500k. The number of servers is eight. In case of 100k, each of the eight server contains at least 10k data. Similarly, for datasets 200k, 300k, 400k and 500k each server has at least 20k, 30k, 40k and 50k data respectively. In this experiment, we keep "s" to 10. Figure 11 shows the results. We observe that the response time increases if the dataset size increases. We also observe that it gradually increases if the dimension increases.

Next, we examine the effect of the number of servers. In each experiment, we distribute 500k data to $m = 2, ..., 8$ servers. In this experiment, we set s = 4 and examine 2D, 3D, 4D and 5D cases. Figure 12 shows the result. We find that if the number of servers increases, the computation time also increases. In the current implementation, we did not consider pipeline execution of the proposed method. It is clear that we can easily parallelize the method by pipeline execution, which is our most significant future work.

6. CONCLUSION

In privacy aware environments, we have to hide individual values and are only allowed to disclose aggregated values of objects. In such situations, skyline query for sets of objects can be a promising alternative in decision-making.

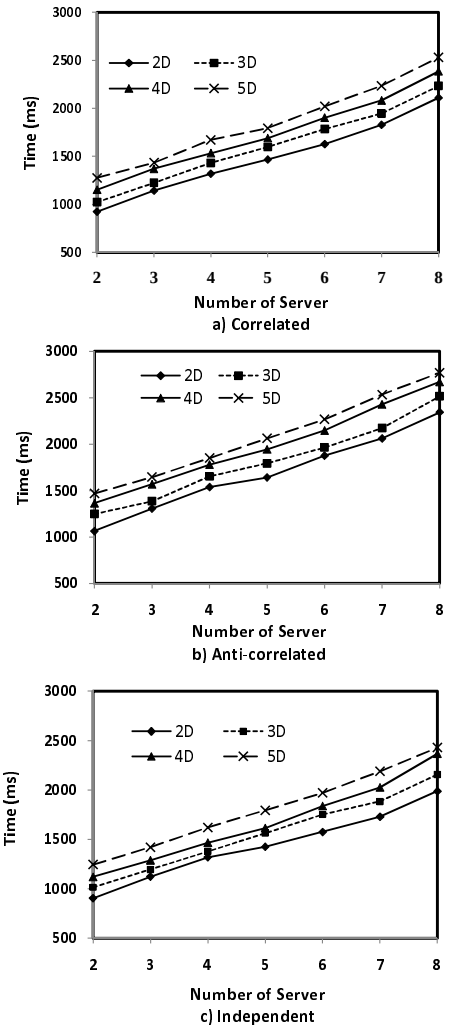


Figure 12: Time Varying The Number of Servers

Moreover, crucial information for decision-making are distributed in different databases in cloud computing environment. Most of the cloud service providers do not want to disclose individual record in their database. Therefore, we present an agent-based skyline set query for a distributed database in this paper. We can compute skyline set query from a distributed database without disclosing individual record to others and can solve the privacy problems of skyline queries in the cloud computing environment.

As we mentioned in the previous section, the proposed method can be parallelized. One of our future direction is to make the proposed method efficient by pipeline executions. Another problem that have to be considered is security enhancement. Currently, we assumed that all participants are honest and do not deceive agents to tap others' information. For example, one server pushes $(0,0,0)$, $(0,0,0)$, and $(-100,100,100)$ for the agent with $\Theta = (-1,0)$ in the example of Figure 6, 7, and 8, intentionally. The server will find the skyline set for $\Theta = (-1,0)$ is $p = (2,6)$ and can predict there is a record $(2,6)$ in another server. We have to consider a protection method against such statistical compromise.

7. ACKNOWLEDGMENTS

This work was partially supported by KAKENHI (19500123). Md. Anisuzzaman Siddique and Md. Shamsul Arefin were supported by the scholarship of MEXT Japan.

8. REFERENCES

[1] S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In Proc. of the IEEE ICDE Conference, pages 421–430, 2001.

[2] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In Proc. of VLDB Conference, pages 275–286, 2002.

[3] C. Li, B. C. Ooi, A. K. H. Tung, and S. Wang. Dada: A data cube for dominant relationship analysis. In Proc. of ACM SIGMOD Conference, pages 659–670, 2006.

[4] Y. Morimoto, T. Fukuda, H. Matsuzawa, K. Yoda, and T. Tokuyama. Algorithms for mining association rules for binary segmentations of huge categorical databases. In Proc. of VLDB Conference, pages 380–391, 1998.

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

[6] J. Pei, W. Jin, M. Ester, and Y. Tao. Catching the best views of skyline: A semantic approach based on decisive subspaces. In Proc. of VLDB Conference, pages 253–264, 2005.

[7] M. A. Siddique and Y. Morimoto. Algorithm for computing convex skyline objectsets on numerical databases. IEICE Trans. on Information and Systems, (10):2709–2716, 2010.

[8] K. L. Tan, P. K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In Proc. of VLDB Conference, pages 301–310, 2001.