Processing of Probabilistic Skyline Queries Using MapReduce

Yoonjae Park
Seoul National University
Seoul, Korea
yjpark@kdd.snu.ac.kr

Jun-Ki Min
Korea Univ. of Tech. & Edu.
Cheonan, Korea
jkmin@kut.ac.kr

Kyuseok Shim
Seoul National University
Seoul, Korea
shim@kdd.snu.ac.kr

ABSTRACT

There has been an increased growth in a number of applications that naturally generate large volumes of uncertain data. By the advent of such applications, the support of advanced analysis queries such as the skyline and its variant operators for big uncertain data has become important. In this paper, we propose the effective parallel algorithms using MapReduce to process the probabilistic skyline queries for uncertain data modeled by both discrete and continuous models. We present three filtering methods to identify probabilistic non-skyline objects in advance. We next develop a single MapReduce phase algorithm *PS-QP-MR* by utilizing space partitioning based on a variant of quadtrees to distribute the instances of objects effectively and the enhanced algorithm *PS-QPF-MR* by applying the three filtering methods additionally. We also propose the workload balancing technique to balance the workload of reduce functions based on the number of machines available. Finally, we present the brute-force algorithms *PS-BR-MR* and *PS-BRF-MR* with partitioning randomly and applying the filtering methods. In our experiments, we demonstrate the efficiency and scalability of *PS-QPF-MR* compared to the other algorithms.

1. INTRODUCTION

There has been an increased growth recently in a number of applications such as social network [1], data integration [15] and sensor data management [13] that naturally produce large volumes of probabilistic/uncertain data. In such applications, uncertainty is inherent due to various factors such as data randomness and incompleteness, limitations of measuring equipments and so on. By the advent of such applications, the support of advanced analysis queries such as the skyline and its variant operators [8, 12, 20] for big uncertain data has become important.

Given a set of certain objects $\mathbb{D} = \{p_1, p_2, \cdots, p_{|\mathbb{D}|}\}$ where each object p_i is represented by a d-dimensional point $\langle p_i(1), p_i(2), \cdots, p_i(d)\rangle$ and $p_i(k)$ is the k-th coordinate of p_i, the skyline is the set of all objects that are not dominated by any other object in $\mathbb{D}$. An object p_i is said to *dominate* an object p_j, denoted by $p_i \prec p_j$, if the following two conditions hold: (1) for every k with $1 \leq k \leq d$, we have $p_i(k) \leq p_j(k)$ and (2) there exists k with $1 \leq k \leq d$ such that $p_i(k) < p_j(k)$.

The notion of the probabilistic skyline was first introduced in [22] for uncertain data. An *uncertain* object can be described by the discrete or continuous uncertainty model. In the discrete model, an object U is modeled as a set of instances and denoted by $U = \{u_1, u_2, \ldots, u_{|U|}\}$ where u_i is a d-dimensional point with its existence probability. In the continuous model, an object U is modeled as an *uncertainty region* with its probabilistic distribution function (pdf).

Given a set of uncertain objects $\mathbb{D}$ represented by the discrete model, a *possible world* is a set of instances from objects in $\mathbb{D}$ where at most a single instance may be selected from each object. The *skyline probability* of an instance is the probability that it appears in a possible world and is not dominated by every instance of the other objects in the possible world. Then, the skyline probability of an object is the sum of the skyline probabilities of its all instances. Similarly, for the continuous model, we define the *skyline probability* of an object by using its uncertainty region and pdf. Given a *probability threshold* T_p, regardless of the uncertainty models used, the probabilistic skyline is the set of uncertain objects whose skyline probabilities are at least T_p.

The modern world is full of devices with sensors and processors. Such deployments of computational resources enable us to measure, collect and process large data from billions of connected devices serving many applications. For example, consider a large number of devices equipped with sensors to measure NO_2 and SO_2 concentrations in the air and are deployed in a wide area to monitor the air pollution. A common characteristics of such sensors is that every measured value is associated with some measurement error, resulting in uncertain data. The pairs of measured NO_2 and SO_2 values by the sensors in each device may be modeled as an object with its uncertainty region and pdf. On the other hand, each device can be also modeled as an object where each pair of measured values can be considered as an instance of the object. To find less polluted locations, we can consider the locations of the devices whose pairs of measured NO_2 and SO_2 values are in the probabilistic skyline. Since these types of applications have the potential to generate a large amount of uncertain data, computing the probabilistic skyline for such big data is challenging today.

Google's MapReduce [11] or its open-source equivalent Hadoop [4] is a powerful and widely used tool that provides easy development of scalable parallel applications such as large-scale graph processing, text processing and machine

Proceedings of the VLDB Endowment, Vol. 8, No. 12

learning to process big data on large clusters of commodity machines. At Google, more than 10,000 distinct programs have been implemented using MapReduce [11].

Many serial algorithms have been proposed to process skyline and its variant queries [8, 12, 20]. Due to the trend of increasing data volumes, several parallel algorithms using MapReduce were developed to process skyline related queries [21, 27] for certain data. Recently, Pei et al. [22] proposed serial algorithms for probabilistic skyline queries. Processing probabilistic skyline queries with MapReduce was also considered in [14] for the special case of discrete model in which each object has *a single instance*.

In this paper, we propose the effective parallel algorithms using MapReduce to process the probabilistic skyline queries for the general case of when each object is expressed by a set of instances in the discrete model or an uncertainty region with its pdf in the continuous model. To the best of our knowledge, processing probabilistic skylines using MapReduce for the general case has not been addressed yet. This paper makes the following contributions:

Early pruning of non-skyline objects: After presenting how to calculate the upper bound of the skyline probability of an object U with a sample of data $\mathbb{D}$, we show that U is not a skyline object if the upper bound is less than the probability threshold T_p. We also demonstrate that we can prune an instance dominated by an instance of another object in a sample of $\mathbb{D}$ in every possible world, after providing how to find such instances. Finally, we present how to maintain and use a small number of objects with high dominating power to reduce the dominance relationship comparisons.

Effective space partitioning by a *PSQtree*: We divide the uncertain objects $\mathbb{D}$ into partitions according to the space split by a variant of a quadtree, called *PSQtree*, which is produced quickly from a sample of $\mathbb{D}$. Since we can identify a leaf node of a *PSQtree* in which every instance can not dominate all instances in another leaf node based on the spatial relationships of regions represented by the leaf nodes, we avoid the useless comparisons between instances in advance. We also present how to exploit the early pruning methods presented previously in the presence of a *PSQtree*.

Implementation on MapReduce: We develop the single MapReduce phase algorithm *PS-QP-MR* by partitioning the space with a *PSQtree*. It calculates the skyline objects in each partition independently by a reduce function. For workload balancing and small transmission overhead, we split data $\mathbb{D}$ into several partitions by a *PSQtree* and cluster them into groups such that (1) the number of groups is a multiple of the number of machines available, (2) the number of objects in every group is similar, (3) the number of instances required to compute the skyline probabilities of the objects in every group is similar and (4) each group is handled by a reduce function with maximum memory usage. We next propose the enhanced algorithm *PS-QPF-MR* by our early pruning methods. We also present the brute-force MapReduce algorithms *PS-BR-MR* and *PS-BRF-MR* based on random partitioning. Finally, by conducting performance study with datasets represented by discrete and continuous uncertainty models, we show that *PS-QPF-MR* is the most effective and scalable algorithm in diverse environments.

2. RELATED WORK

Skyline processing was first investigated in the context of databases in [8]. Dynamic and reverse skylines were introduced in [20] and [12], respectively. In addition, several algorithms have been proposed for skyline queries in [16, 20, 24]. Kossman et al. [16] proposed a Nearest Neighbor (NN) approach to retrieve skylines. Papadias et al. [20] improved the NN algorithm by using the branch-and-bound (BBS) strategy. Many existing algorithms utilize R*-trees [6] to check dominance relationships between points.

Due to the importance of supporting applications dealing with uncertain data, the techniques for processing uncertain queries such as probabilistic top-K [23] and similarity join [18] queries have been proposed. Refer to [26] for the summary of processing uncertain queries.

The serial algorithms for probabilistic skyline processing over uncertain data have been introduced in [5, 22]. The skyline probabilities of all objects in the discrete model are computed without considering the minimum probability threshold in [5]. Skyline computation with the minimum probability threshold is considered in [22] for both discrete and continuous models, but every instance of each object has the same existence probability. To parallelize such serial algorithms, we need two MapReduce phases. The first phase splits data into partitions randomly and computes the partial skyline probabilities of every object in each partition independently. The second phase computes the skyline probability of each object by collecting its partial skyline probabilities from different partitions. In this paper, we address a generalized problem of both [5] and [22], and we compute the probabilistic skylines with the minimum probability threshold for the discrete and continuous models.

Recently, parallel skyline processing algorithms with MapReduce for certain and uncertain data were presented in [2, 21, 27] and [14], respectively. We can develop the parallel algorithms for uncertain data by simply performing one of the algorithms for certain data in [2, 21, 27] for every possible world. However, due to a lot of possible worlds, naive extensions of such algorithms to uncertain data are very inefficient and impractical. The most relevant work to ours is the MapReduce algorithm *PSMR* [14], but *PSMR* can compute the probabilistic skylines only for the case where each uncertain object has a single instance in the discrete model.

3. PRELIMINARIES

We first introduce the definition of the probabilistic skyline [22] by the popular *possible worlds* semantics [3, 10] and next present the state-of-the-art in [14].

3.1 Probabilistic Skylines

The discrete model: Given a set of uncertain objects $\mathbb{D}$, an object $U \in \mathbb{D}$ is modeled as a set of instances and denoted by $U = \{u_1, u_2, \cdots, u_{|U|}\}$ where u_i is associated with an existence probability $P(u_i)$ such that $\sum_{u_i \in U} P(u_i) \leq 1$. A *possible world* is a materialized set of instances from objects. Since all instances of U are mutually exclusive, multiple instances of U cannot belong to a possible world simultaneously. The probability that an instance $u_i \in U$ appears in a possible world is $P(u_i)$ and the probability that any instance of an object U does not appear is $1 - \sum_{u_i \in U} P(u_i)$.

When a possible world contains an instance $u_i \in U$, if any instance v_j of every other object $V \in \mathbb{D}$ dominating u_i does not exist in the possible world, u_i is a skyline instance in the possible world. Since such a probability is $\prod_{V \in \mathbb{D}, V \neq U}(1 - \sum_{v_j \in V, v_j \prec u_i} P(v_j))$, the skyline probability of u_i, denoted by $P_{sky}(u_i)$, can be written as follows [5]:

$$P_{sky}(u_i) = P(u_i) \times \prod_{V \in \mathbb{D}, V \neq U} (1 - \sum_{v_j \in V, v_j \prec u_i} P(v_j)). \quad (1)$$

We define the skyline probability of an object U, denoted by $P_{sky}(U)$, as the sum of the skyline probabilities of all its instances (i.e., $P_{sky}(U) = \sum_{u_i \in U} P_{sky}(u_i)$).

The continuous model: An uncertain object $U \in \mathbb{D}$ is modeled as an *uncertainty region* $U.R$ with its probabilistic distribution function $U.f(\cdot)$ [7, 17, 22]. We assume that each uncertainty region is a hyper-rectangle as in [17]. The probability that an instance of U is located at a point u in $U.R$ is $U.f(u)$ where $\int_{U.R} U.f(u)\mathrm{d}u = 1$.

Given an object $U \in \mathbb{D}$, $P_{sky}(U)$ is defined in [22] as:

$$P_{sky}(U) = \int_{U.R} U.f(u) \prod_{V \in \mathbb{D}, V \neq U} (1 - \int_{V.R} V.f(v)\mathbb{1}(v \prec u)\mathrm{d}v)\mathrm{d}u \quad (2)$$

where $\mathbb{1}(v \prec u)$ is an indicator function which returns 1 if v dominates u, and 0 otherwise.

DEFINITION 3.1: **[Probabilistic Skyline Problem]** *For a set of uncertain objects $\mathbb{D}$ and a probability threshold T_p, the probabilistic skyline, denoted by $pSL(\mathbb{D}, T_p)$, is the set of all objects whose skyline probabilities are at least T_p. That is, $pSL(\mathbb{D}, T_p) = \{U \in \mathbb{D} \mid P_{sky}(U) \geq T_p\}$.*

EXAMPLE 3.2: *Consider a set of objects $\mathbb{D} = \{W, X, Y, Z\}$ with the discrete model in Figure* 1(a)*. Each instance of objects represents a pair of measured values of NO_2 and SO_2 concentrations. In Figure* 1(b)*, we plot every instance in $\mathbb{D}$ into a 2-dimensional space. Since y_1 is dominated by w_1, w_2, x_1, x_2 and z_2, the skyline probability of y_1 computed by Equation* (1) *is $P_{sky}(y_1)$=$P(y_1)(1-P(w_1)-P(w_2))(1-P(x_1)-P(x_2))(1-P(z_2))$=0.024. Similarly, $P_{sky}(y_2)$=0.012. The skyline probability of Y is $P_{sky}(Y)$=$P_{sky}(y_1)$+$P_{sky}(y_2)$=0.036. Furthermore, $P_{sky}(W)$=0.9, $P_{sky}(X)$=0.4 and $P_{sky}(Z)$=0.74. When T_p is* 0.5*, $pSL(\mathbb{D}, T_p)$ is $\{W, Z\}$ by Definition* 3.1. ∎

3.2 PSMR: The State-of-the-art Algorithm

The algorithm *PSMR* in [14] works with two MapReduce phases as follows.

The first phase: It computes the local candidate and affect sets. The candidate set contains possible probabilistic skyline objects and the affect set includes the probabilistic non-skyline objects required to compute the skyline probabilities of the objects in the candidate set. The first statement in Lemma 3.3 is used to find the candidate skyline objects. Then, it applies the second statement in Lemma 3.3 to discover the affect set among probabilistic non-skyline objects. Note that $\mathbb{S}$ in the following lemma represents the set of objects to be processed in each machine.

LEMMA 3.3: [14] *For a set $\mathbb{S} \subset \mathbb{D}$, if an object $U = \{u\}$ in $\mathbb{D}$ satisfies $P(u)\prod_{V \in \mathbb{S}}(1-\sum_{v \in V, v \prec u} P(v)) < T_p$, it is a probabilistic non-skyline object in $\mathbb{D}$ (i.e., $U \notin pSL(\mathbb{D}, T_p)$). Furthermore, if an object $U = \{u\}$ satisfies $\prod_{V \in \mathbb{S}}(1-\sum_{v \in V, v \prec u} P(v)) < T_p$, it does not belong to the affect set.*

At the end of the first phase, *PSMR* generates the candidate and affect sets by combining the local candidate and affect sets produced by all machines, respectively.

The second phase: *PSMR* first divides the union of the candidate and affect sets into several partitions each of which is allocated to a different machine. After broadcasting the candidate set to every machine, each machine computes the partial skyline probabilities of all broadcast candidate objects by using the objects in its allocated partition. Then, they gather all partial skyline probabilities of each object from different machines into one of the machines to calculate the skyline probabilities of all candidate objects in parallel.

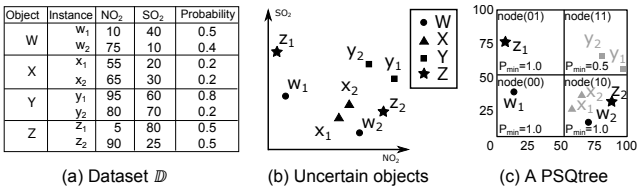


Figure 1: An example of an uncertain dataset

The second phase is necessary in *PSMR* to compare every candidate object with all other objects in the candidate and affect sets. In contrast, our proposed algorithms *PS-QP-MR* and *PS-QPF-MR* consist of a *single* MapReduce phase.

4. KEY IDEAS OF OUR ALGORITHMS

4.1 Early Pruning Techniques

If we know that an object cannot be a probabilistic skyline object, we can avoid computing its skyline probability.

4.1.1 Upper-bound Filtering

The following propositions address that the skyline probability of an object U by considering a sample $\mathbb{S}$ of the objects in $\mathbb{D}$ only is an *upper bound* of $P_{sky}(U)$ for both discrete and continuous models.

PROPOSITION 4.1: *Consider an object U in the discrete model. For an instance $u_i \in U$, the value of $P_{sky}(u_i)$ computed by Equation* (1) *with $V' \subseteq V$ and $\mathbb{S} \subset \mathbb{D}$ instead is the* upper bound *of $P_{sky}(u_i)$. The sum of the upper bounds of $P_{sky}(u_i)$s with all $u_i \in U$ is the* upper bound *of $P_{sky}(U)$.*

PROPOSITION 4.2: *Consider an object U modeled by its uncertainty region $U.R$ with a pdf $U.f(\cdot)$. The value of $P_{sky}(U)$ computed by Equation* (2) *with $\mathbb{S} \subset \mathbb{D}$ and a sub-region $V.R'$ of $V.R$ becomes the* upper bound *of $P_{sky}(U)$.*

By keeping the upper bounds of the skyline probabilities of all instances in each object, we can identify probabilistic non-skyline objects. As shown in Figure 1(c), all instances in $\langle[50, 100), [50, 100)\rangle$ are dominated by w_1. Note that $P_{sky}(y_1)$=$P(y_1)\prod_{V \in \mathbb{D}, V \neq Y}$ (1-$\sum_{v_j \in V, v_j \prec y_1} P(v_j)$)=0.024 by Equation (1). Due to Proposition 4.1, we have $P_{sky}(y_1) \leq P(y_1)$(1-$P(w_1)$)=0.4 which is obtained by using $\mathbb{S} = \{W\}$ and $V' = \{w_1\}$. Similarly, the upper bound of $P_{sky}(y_2)$ becomes 0.1. Thus, the upper bound of $P_{sky}(Y)$ becomes 0.5 by adding the upper bounds of $P_{sky}(y_1)$ and $P_{sky}(y_2)$. If T_p is 0.6, since $P_{sky}(Y) \leq 0.5 < T_p$, Y is a non-skyline object.

We now present how to compute the upper bound of the skyline probability of every object in each partition for our upper-bound filtering. Let $R.min$ be the point whose k-th coordinate is the minimum in the k-th dimension for a rectangular region R.

DEFINITION 4.3: *For an instance u_i of an object $U \in \mathbb{D}$, a set of objects $\mathbb{S} \subset \mathbb{D}$ and a rectangular region $R(u_i)$ including u_i, we define*

$$\beta(U, \mathbb{S}, R(u_i)) = \frac{\prod_{V \in \mathbb{S}}(1 - \sum_{v_j \in V, v_j \prec R(u_i).min} P(v_j))}{1 - \sum_{u_k \prec R(u_i).min, u_k \in U} P(u_k)} \quad (3)$$

$$\text{and } up(u_i, U, \mathbb{S}, R(u_i)) = P(u_i) \times \beta(U, \mathbb{S}, R(u_i)).$$

The upper bound of the skyline probability of an instance can be computed by utilizing the following lemma.

LEMMA 4.4: *Consider an instance u_i of an object $U \in \mathbb{D}$ and a rectangular region $R(u_i)$ which contains u_i. For a set of objects $\mathbb{S} \subset \mathbb{D}$, we have $P_{sky}(u_i) \leq up(u_i, U, \mathbb{S}, R(u_i))$.*

Proof: Let D_{u_i} and D_R be the sets of the instances in $\mathbb{D}$ which dominate u_i and $R(u_i).min$, respectively. Since every instance dominating $R(u_i).min$ also dominates u_i, we have $D_R \subseteq D_{u_i}$. We derive $P_{sky}(u_i) \leq up(u_i, U, \mathbb{S}, R(u_i))$ as follows:

$$P_{sky}(u_i) \leq P(u_i) \prod_{V \in \mathbb{S}, V \neq U} \left(1 - \sum_{v_j \in V \cap D_R} P(v_j)\right) \text{ (by Proposition 4.1)}$$

$$= P(u_i) \prod_{V \in \mathbb{S}, V \neq U} \left(1 - \sum_{v_j \in V \cap D_R} P(v_j)\right) \frac{1 - \sum_{u_i \in U \cap D_R} P(u_i)}{1 - \sum_{u_i \in U \cap D_R} P(u_i)}$$

$$\leq P(u_i) \frac{\prod_{V \in \mathbb{S}} \left(1 - \sum_{v_j \in V \cap D_R} P(v_j)\right)}{1 - \sum_{u_k \in U \cap D_R} P(u_k)}$$

$$= P(u_i) \frac{\prod_{V \in \mathbb{S}} (1 - \sum_{v_j \in V, v_j \prec R(u_i).min} P(v_j))}{1 - \sum_{u_k \prec R(u_i).min, u_k \in U} P(u_k)} = up(u_i, U, \mathbb{S}, R(u_i))$$

(since $\{v_j \in V | v_j \prec R(u_i).min\} = V \cap D_R$) ∎

COROLLARY 4.5: *Consider an object $U \in \mathbb{D}$ and let $R(u_i)$ be a rectangular region containing an instance u_i of U. For a set of objects $\mathbb{S} \subset \mathbb{D}$, if we have $\sum_{u_i \in U} up(u_i, U, \mathbb{S}, R(u_i)) < T_p$, U is not a probabilistic skyline object.*

By Corollary 4.5, we do not compute the skyline probability of an uncertain object U if we have $\sum_{u_i \in U} up(u_i, U, \mathbb{S}, R(u_i)) < T_p$. We call such pruning the *upper-bound filtering*. We can prune even further when every object has a single instance only as follows.

LEMMA 4.6: *When every object in $\mathbb{D}$ has a single instance, consider an instance u of an object U and a rectangular region $R(u)$ containing u. For a set of objects $\mathbb{S} \subset \mathbb{D}$, if we have $up(u, U, \mathbb{S}, R(u)) = P(u) \times \beta(U, \mathbb{S}, R(u)) < T_p$, U is not a skyline object. Furthermore, if $\beta(U, \mathbb{S}, R(u)) < T_p$ also holds, there is no object in the probabilistic skyline whose instance is dominated by u.*

Proof: Since every object has a single instance, we have $P_{sky}(U) = P_{sky}(u) \leq up(u, U, \mathbb{S}, R(u)) \leq T_p$ by Lemma 4.4 and U is not a skyline object due to Corollary 4.5.

We next prove the second case of when $\beta(U, \mathbb{S}, R(u)) < T_p$ by contradiction. Assume that there is a skyline object W whose instance w is dominated by u (i.e., $u \prec w$). By Proposition 4.1, we have $P_{sky}(W) \leq P(w) \prod_{V \in \mathbb{S}} (1 - \sum_{v \in V, v \prec w} P(v))$. Since $R(u)$ contains u, $R(u).min \prec u \prec w$ holds and hence $R(u).min \prec w$. Furthermore, because every instance v such that $v \prec R(u).min$ also dominates w, we have $\{v \in V | v \prec R(u).min\} \subseteq \{v \in V | v \prec w\}$ and get

$$P(w) \prod_{V \in \mathbb{S}} (1 - \sum_{v \in V, v \prec w} P(v)) \leq P(w) \prod_{V \in \mathbb{S}} (1 - \sum_{v \in V, v \prec R(u).min} P(v)).$$

When U has a single instance u which does not dominate $R(u).min$, we have $\sum_{u \prec R(u).min, u \in U} P(u) = 0$ resulting that $\beta(U, \mathbb{S}, R(u)) = \prod_{V \in \mathbb{S}} (1 - \sum_{v \in V, v \prec R(u).min} P(v))$ from Definition 4.3. Thus, we have $P_{sky}(W) \leq P(w) \times \beta(U, \mathbb{S}, R(u))$. Now assume that $\beta(U, \mathbb{S}, R(u)) < T_p$ holds. Then, we obtain $P_{sky}(W) < T_p$. It contradicts to the assumption that W is a skyline object. ∎

The following corollary shows that the Lemma 3.3 used by *PSMR* [14] is a special case of our Lemma 4.6 since $\beta(U, \mathbb{S}, R(u)) = \prod_{V \in \mathbb{S}} (1 - \sum_{v_j \in V, v_j \prec u} P(v_j))$ when $R(u)$ degenerates to the minimum bounding rectangle containing only a single instance u.

COROLLARY 4.7: *When every object in $\mathbb{D}$ has a single instance, consider an instance u of an object U and a subset $\mathbb{S} \subset \mathbb{D}$. If $P(u) \prod_{V \in \mathbb{S}} (1 - \sum_{v_j \in V, v_j \prec u} P(v_j)) < T_p$, U is not a skyline object. Furthermore, when $\beta(U, \mathbb{S}, R) = \prod_{V \in \mathbb{S}} (1 - \sum_{v_j \in V, v_j \prec u} P(v_j)) < T_p$ also holds, there is no object in the probabilistic skyline whose instance is dominated by u.*

The continuous model: We define $up_{pdf}(u, U, \mathbb{S}, R(u))$ by replacing the summations in Definition 4.3 with integrations over all points contained in $V.R$ for every object $V \in \mathbb{S}$.

DEFINITION 4.8: *For an object $U \in \mathbb{D}$ with its uncertainty region $U.R$, a point u located in $U.R$, a subset $\mathbb{S} \subset \mathbb{D}$ and a rectangular region $R(u)$ which contains u, $up_{pdf}(u, U, \mathbb{S}, R(u))$ is defined as follows:*

$$up_{pdf}(u, U, \mathbb{S}, R(u)) = U.f(u) \frac{\prod_{V \in \mathbb{S}} (1 - \int_{V.R} V.f(v) \mathbb{1}(v \prec R(u).min) dv)}{1 - \int_{U.R} U.f(w) \mathbb{1}(w \prec R(u).min) dw}$$

where $R(u).min$ is the point whose k-th coordinate is the minimum in the k-th dimension for $R(u)$.

The following lemma states the condition of when an object is not a probabilistic skyline object. Since the proof is similar to that of Lemma 4.4, we omit it.

LEMMA 4.9: *Consider the skyline threshold T_p, an object $U \in \mathbb{D}$ and a point u in $U.R$. Let $R(u)$ be a rectangular region containing u. For an object $U \in \mathbb{D}$ and a subset $\mathbb{S} \subset \mathbb{D}$, when $\int_{U.R} up_{pdf}(u, U, \mathbb{S}, R(u)) du < T_p$, U is not a skyline object.*

4.1.2 Zero-probability Filtering

Recall that the skyline probability of $u_i \in U$ is $P_{sky}(u_i) = P(u_i) \prod_{V \in \mathbb{D}, V \neq U} (1 - \sum_{v_j \in V, v_j \prec u_i} P(v_j))$. When $P_{sky}(u_i) = 0$, there exists an object V such that $\sum_{v_j \in V, v_j \prec u_i} P(v_j) = 1$. Thus, an instance of V dominating u_i always appears in every possible world and u_i cannot contribute to computing the skyline probability of every other object.

LEMMA 4.10: *Consider an instance u_i of an object $U \in \mathbb{D}$ and a rectangular region $R(u_i)$ containing u_i. For a subset $\mathbb{S} \subset \mathbb{D}$, when $\prod_{V \in \mathbb{S}} (1 - \sum_{v_j \in V, v_j \prec R(u_i).min} P(v_j)) = 0$, the skyline probability of u_i is zero and we can delete u_i from U.*

Proof: By Lemma 4.4, we have $P_{sky}(u_i) \leq up(u_i, U, \mathbb{S}, R(u_i)) = P(u_i) \times \beta(U, \mathbb{S}, R(u_i))$. If $\prod_{V \in \mathbb{S}} (1 - \sum_{v_j \in V, v_j \prec R(u_i).min} P(v_j)) = 0$ (i.e., the numerator in Equation (3) of $\beta(U, \mathbb{S}, R(u_i))$ is zero), we have $0 \leq P_{sky}(u_i) \leq up(u_i, U, \mathbb{S}, R(u_i)) = 0$. ∎

We refer to the pruning technique based on Lemma 4.10 as the *zero-probability filtering*.

The continuous model: When $up_{pdf}(u, U, \mathbb{S}, R(u)) = 0$ holds for all $u \in U.R$, we have $\int_{U.R} up_{pdf}(u, U, \mathbb{S}, R(u)) du = 0$ and U is not a skyline object by Lemma 4.9. Thus, we can delete U.

4.1.3 Dominance-Power Filtering

We maintain a small number of objects with the high dominating power and use them for checking the dominance relationship to handle large data.

DEFINITION 4.11: *Consider a d-dimensional space $\langle [0, b(1)), \cdots, [0, b(d)) \rangle$ where $[0, b(k))$ is its range of the k-th dimension. The* dominating power *of an instance $v_j = \langle v_j(1), \cdots, v_j(d) \rangle$, denoted by $DP(v_j)$, is $\prod_{k=1}^{d} (b(k) - v_j(k))$. Furthermore, the dominance power of an object V, denoted by $DP(V)$ is $\sum_{v_j \in V} (P(v_j) \times DP(v_j))$.*

As the existence probability of an instance v_j of an object V increases, the skyline probability of u_i of another object U dominated by v_j decreases. In addition, the number of instances of other objects dominated by v_j tends to be larger as the dominating power $DP(v_j)$ grows. Thus, we utilize $DP(V)$ to estimate the dominating power of V. We refer to the set of top-K objects with the largest dominating powers as a *dominating object set* F.

For an object U with a dominating object set F, if we have $\sum_{u_i \in U} P(u_i) \prod_{V \in F, V \neq U} (1 - \sum_{v_j \in V, v_j \prec u_i} P(v_j)) < T_p$, U is not a probabilistic skyline object in $\mathbb{D}$ and thus we do not compute its skyline probability. We call the strategy the *dominance-power filtering*.

To maintain the K objects with the largest dominating powers and identify non-skyline objects at the same time, we invoke the procedure *DP-Filter* which utilizes a min-heap H to store the K dominating objects. For an object U, if the value of $P_{sky}(U)$ which is computed by considering H instead of $\mathbb{D}$ is less than T_p, *DP-Filter* returns FALSE to indicate that U is not a probabilistic skyline object due to Proposition 4.1. Otherwise, it returns TRUE. In this case, we also update H by inserting U. In other words, if the number of objects in H is less than K, we insert the object U into H. When the number of objects in H is K and the dominance power of U is larger than that of the object O with the minimum dominance power in H, we delete O from H and insert U to H.

The continuous model: Consider a d-dimensional space $\langle[0,b(1)),\cdots,[0,b(d))\rangle$. The dominance power of an object U in the continuous model, represented by $DP_{pdf}(U)$, is defined as $\int_{U.R} U.f(u) \prod_{k=1}^{d} (b(k) - u(k)) \mathrm{d}u$. We keep top-$K$ objects with the highest dominating powers as the dominating object set F. The only change is to utilize Proposition 4.2 instead of Proposition 4.1. If $\int_{U.R} U.f(u) \prod_{V \in F, V \neq U} (1 - \int_{V.R} V.f(v) \mathbb{1}(v \prec u) \mathrm{d}v) \mathrm{d}u < T_p$ holds for an object U, *DP-Filter* returns FALSE. Otherwise, it returns TRUE and update H with U.

4.2 Utilization of a PSQtree for Pruning

To divide the data space into several sub-spaces, we develop a variant of *sky-quadtrees* in [21], called the *PSQtree*.

4.2.1 Generating a PSQtree

We recursively divide d-dimensional space into equi-sized 2^d sub-spaces, each of which is associated with a node in a *PSQtree*, until the number of points in each sub-space does not exceed the *split threshold*, denoted by ρ. We refer to the region represented by a node n as $n.region = \langle[n(1)^-, n(1)^+), \cdots, [n(d)^-, n(d)^+)\rangle$ where $[n(k)^-, n(k)^+)$ is the k-th dimensional range. We also define $n.min$ ($n.max$) as the $n.region$'s closest (farthest) corner of a leaf node n from the origin. Each node n is assigned with an id according to the method in [21] and the node with an id "i" is represented by $node(i)$. To build a *PSQtree* quickly, we utilize a random sample $\mathbb{S}$ of the objects in $\mathbb{D}$. Figure 1(c) shows an example of a *PSQtree* produced by the subset $\mathbb{S} = \{W, Z\}$ of $\mathbb{D}$ in Figure 1(a).

4.2.2 Exploiting a *PSQtree* for Filtering

In this section, we show how the filtering techniques presented previously can be exploited by using a *PSQtree*.

DEFINITION 4.12: *Consider a dataset $\mathbb{D}$, and a leaf node n of a PSQtree built by a sample $\mathbb{S} \subset \mathbb{D}$. We define $n.P_{min}(\mathbb{S}) = \prod_{V \in \mathbb{S}} (1 - \sum_{v_j \in V, v_j \prec n.min} P(v_j))$ for the discrete model and $n.P_{min}(\mathbb{S}) = \prod_{V \in \mathbb{S}} (1 - \int_{V.R} V.f(v) \mathbb{1}(v \prec n.min) dv)$ for the continuous model.*

By traversing the *PSQtree*, we set $n.P_{min}(\mathbb{S})$ in each leaf node n where $\mathbb{S}$ is the sample used to build the *PSQtree* and initially $n.P_{min}(\mathbb{S})$=1. In each leaf node n, we scan every object $V \in \mathbb{S}$ to check whether $n.min$ is dominated by an instance v_j of V and compute the sum of $P(v_j)$ of every instance v_j dominating $n.min$. We next update $n.P_{min}(\mathbb{S})$ by multiplying $(1 - \sum_{v_j \in V, v_j \prec n.min} P(v_j))$ to itself according to Definition 4.12. For the continuous model, we generate the points in $V.R$ for each object $V \in \mathbb{S}$ by following $V.f(\cdot)$ and build a *PSQtree* by using the generated points.

Upper-bound filtering: We can utilize $n.P_{min}(\mathbb{S})$ for the upper-bound filtering due to the following corollary. The proof of the corollary is analogous to that of Lemmas 4.4 and 4.9 by letting $R(u_i).min = n(u_i).min$.

COROLLARY 4.13: *For a PSQtree T generated by a sample $\mathbb{S} \subset \mathbb{D}$ and an instance u_i of an object U, let $n(u_i)$ be the leaf node of T whose region contains u_i. Depending on an uncertainty model, the skyline probability of U (i.e., $P_{sky}(U)$) is upper bounded by $up_T(U, \mathbb{S})$ where*

$$up_T(U,\mathbb{S}) = \begin{cases} \sum_{u_i \in U} \frac{P(u_i) \times n(u_i).P_{min}(\mathbb{S})}{1 - \sum_{u_k \prec n(u_i).min, u_k \in U} P(u_k)} \\ \int_{U.R} \frac{U.f(u_i) \times n(u_i).P_{min}(\mathbb{S})}{1 - \int_{U.R} U.f(w) \mathbb{1}(w \prec n(u_i).min) dw} \mathrm{d}u_i. \end{cases}$$

Zero-probability filtering: We also use $n.P_{min}(\mathbb{S})$ for the zero-probability filtering by the following corollary whose proof is similar to that of Lemma 4.10.

COROLLARY 4.14: *For a leaf node n of a PSQtree built by a sample $\mathbb{S} \subset \mathbb{D}$, when $n.P_{min}(\mathbb{S})$=0, the skyline probability of every instance in the $n.region$ is zero and thus we can delete the instances of all objects in the $n.region$ from $\mathbb{D}$.*

To build a *PSQtree*, the procedure *GenQtree* is called with a sample $\mathbb{S}$ of the objects in $\mathbb{D}$. We omit the pseudocode of *GenQtree* since it is straightforward.

4.2.3 Partitioning Objects by a PSQtree

For an object $U \in \mathbb{D}$, if we distribute its instances to several partitions, we need an additional aggregation phase to compute the skyline probability of U by summing the skyline probabilities of its instances in multiple partitions. To guarantee that the skyline probability of each object can be computed without an extra MapReduce phase, we allocate all instances of each object U to a single partition by utilizing $U.max$ defined as follows.

DEFINITION 4.15: *For the discrete model, the* max *and* min *points of an object U, represented by $U.max$ and $U.min$, are defined as $U.max(k) = max_{u_i \in U} u_i(k)$ and $U.min(k) = min_{u_i \in U} u_i(k)$, respectively, for $k = 1, \ldots, d$. For the continuous model, where U is modeled by an uncertainty region $U.R$ with pdf, $U.max$ ($U.min$) is the farthest (closest) corner point in $U.R$ from the origin.*

Let $\mathcal{M}(\mathbb{D}, n_\ell)$ be the set of objects whose max points belong to a leaf node n_ℓ of a *PSQtree*. We need to identify all the other objects required to compute the skyline probability of every object $U \in \mathcal{M}(\mathbb{D}, n_\ell)$. To do this efficiently, we use the dominance relationship between a pair of leaf nodes.

DEFINITION 4.16: *For a pair of nodes n_1 and n_2 in a PSQtree, if $n_1.min(k) < n_2.max(k)$ for $k = 1, \cdots, d$, we say n_1* weakly dominates *n_2 and represent it by $n_1 \preceq n_2$.*

```
Function PS-QPF-MR(𝔻, T_p, ρ)
𝔻: uncertain dataset, T_p: probability threshold, ρ: split threshold
begin
1.  𝕊 = Sample(𝔻);
2.  PSQtree = GenQtree(𝕊, ρ);
3.  Broadcast PSQtree; Broadcast T_p;
4.  pSL = RunMapReduce(PS-QPFC-MR, 𝔻);
5.  return pSL;
end
```

Figure 2: The PS-QPF-MR algorithm

Consider the *PSQtree* in Figure 1(c). The min point of $node(00)$ (i.e., $node(00).min$) is $\langle 0, 0\rangle$ and the max point of $node(11)$ (i.e., $node(11).max$) is $\langle 100, 100\rangle$. Since $node(00).min(1) < node(11).max(1)$ and $node(00).min(2) < node(11).max(2)$, $node(00) \preceq node(11)$. We also have $node(00).min(k) < node(01).max(k)$ for every k and $node(00) \preceq node(01)$. However, since $node(01).min(2) \geq node(10).max(2)$, $node(01)$ does not weakly dominate $node(10)$.

For each leaf node n_ℓ, Lemma 4.17 shows that the exact skyline probabilities of the objects in $\mathcal{M}(\mathbb{D}, n_\ell)$ can be computed by considering only the instances located in the region of every leaf node which weakly dominates n_ℓ in both discrete and continuous models.

LEMMA 4.17: *Consider a dataset $\mathbb{D}$, a leaf node n_ℓ of a PSQtree and an object $U \in \mathcal{M}(\mathbb{D}, n_\ell)$. In the discrete model, for each instance u_i of U, if an instance v_j of another object $V \in \mathbb{D}$ is contained in the region of a leaf node n such that $n \not\preceq n_\ell$, v_j does not dominate u_i. In the continuous model, if $V.R.min$ does not dominate $n_\ell.max$ for another object $V \in \mathbb{D}$, V does not affect the skyline probability of U.*

Proof: Since the object U is in $\mathcal{M}(\mathbb{D}, n_\ell)$, $U.max$ is contained in $n_\ell.region$. Consider the discrete model first. For an instance $u_i \in U$, $u_i(k) \leq U.max(k) < n_\ell.max(k)$ holds for $k = 1, \cdots, d$. Since $n \not\preceq n_\ell$, there exists a value k such that $n_\ell.max(k) \leq n.min(k)$. Because v_j is contained in $n.region$, we have $n.min(k) \leq v_j$. Thus, we have $u_i(k) < n_\ell.max(k) \leq n.min(k) \leq v_j$ and v_j does not dominate u_i. Similarly, we can prove the case of the continuous model. ∎

According to Lemma 4.17, we define the set of instances of an object $V \notin \mathcal{M}(\mathbb{D}, n_\ell)$ required to compute the skyline probability of every object U in $\mathcal{M}(\mathbb{D}, n_\ell)$.

DEFINITION 4.18: *For a leaf node n_ℓ, let $\mathcal{I}_w(\mathbb{D}, n_\ell)$ be all instances of an object in $\mathbb{D} - \mathcal{M}(\mathbb{D}, n_\ell)$ which are in a leaf node n satisfying $n \preceq n_\ell$. In other words, $\mathcal{I}_w(\mathbb{D}, n_\ell) = \{v_j \in V | V \notin \mathcal{M}(\mathbb{D}, n_\ell) \wedge n(v_j) \preceq n_\ell\}$.*

Consider the dataset $\mathbb{D}$ and the *PSQtree* in Figure 1. $\mathcal{I}_w(\mathbb{D}, node(10))$ is $\{w_1, w_2, z_2\}$ since $node(00)$ and $node(10)$ weakly dominate $node(10)$ as well as $\mathcal{M}(\mathbb{D}, node(10)) = \{X\}$.

5. MAPREDUCE ALGORITHMS WITH QUADTREE PARTITIONING

In this section, we develop the algorithms with a single MapReduce phase by distributing the objects based on the space split by a *PSQtree*.

5.1 PS-QPF-MR: The Algorithm with Quadtree Partitiong and Filtering

We first present the MapReduce algorithm *PS-QP-MR* (Probabilistic Skyline algorithm by Quadtree Partitioning) which utilizes a *PSQtree*. Then, we provide the MapReduce algorithm *PS-QPF-MR* which enhances *PS-QP-MR* by applying the filtering techniques described in Section 4.

PS-QP-MR: We build a *PSQtree* with a sample $\mathbb{S}$ of data $\mathbb{D}$ in a single machine by calling *GenQtree* introduced in Section 4.2. We next split $\mathbb{D}$ using MapReduce into partitions each of which corresponds to a leaf node n_ℓ of the *PSQtree* and contains the objects in $\mathcal{M}(\mathbb{D}, n_\ell)$ as well as the instances in $\mathcal{I}_w(\mathbb{D}, n_\ell)$ (by Definition 4.18). We then compute the skyline probability of each object U in $\mathcal{M}(\mathbb{D}, n_\ell)$ and output U if U is a probabilistic skyline object.

```
Function PS-QPFC-MR.setup()
begin
1.  H = InitMinHeap(); PSQtree = LoadPSQtree();
end
Function PS-QPFC-MR.map(U)
U: an uncertain object
begin
1.  T_p = LoadThreshold();
2.  U' = ZeroProb(U, PSQtree);
3.  upper = UpperBound(U', PSQtree);
4.  cand = FALSE;
5.  if upper ≥ T_p then
6.      cand = DP-Filter(U', T_p, H);
7.      if cand then emit(n(U'.max), (U', 'C'));
8.  for each leaf node n_ℓ in PSQtree do
9.      if cand = True and n_ℓ = n(U'.max) then continue;
10.     I = NewList();
11.     for each u_i in U' do
12.         if n(u_i) ⪯ n_ℓ then
13.             I.add(u_i));
14.     emit(n_ℓ, (I, 'W', cand))
end
Function PS-QPFC-MR.reduce(n_ℓ, L)
begin
1.  (L_C, L_W^T, L_W^F) = SplitList(L);
2.  T_p = LoadThreshold();
3.  for each object U in L_C do
4.      skyline_prob = SkylineProb(U, L_C, L_W^T, L_W^F);
5.      if skyline_prob ≥ T_p then
6.          emit( U, skyline_prob );
end
```

Figure 3: The PS-QPFC-MR algorithm

PS-QPF-MR: The only difference of *PS-QPF-MR* from *PS-QP-MR* is to check whether each object U is a skyline candidate object or not by using the three filtering techniques and to compute the skyline probabilities of only skyline candidate objects. We present the pseudocode of *PS-QPF-MR* in Figure 2 and that of the procedure *PS-QPFC-MR* called by *PS-QPF-MR* in Figure 3 (line 4 in Figure 2).

Setup function: Before map functions are called, the setup function of each mapper task initializes a min-heap H and loads a *PSQtree* to share them across the map functions. The min-heap H maintains the dominating object set F for the dominance-power filtering introduced in Section 4.1.3.

Map function: The map function invoked with an object U loads the probability threshold T_p (line 1 of PS-QPFC-MR.map). We apply the zero-probability, upper-bound and dominance-power filtering techniques by invoking *ZeroProb*, *UpperBound* and *DP-Filter*, respectively (lines 2-6). We refer to U' as the object after pruning U's instances by *ZeroProb*. If the upper bound of the $P_{sky}(U')$ computed by *UpperBound* is at least T_p, *DP-Filter* is invoked to check whether U' is a candidate object or not. If U' is a candidate object (i.e., *cand* =TRUE), the map function emits the key-value pair $\langle n(U'.max), (U', \text{'C'})\rangle$ where $n(U'.max)$ is the leaf node containing $U'.max$ and 'C' represents that U' is a skyline candidate contained in $\mathcal{M}(\mathbb{D}, n(U'.max))$ (line 7).

For each leaf node n_ℓ, we emit each instance u_i of U' which is required to compute the exact skyline probabilities of objects in $\mathcal{M}(\mathbb{D}, n_\ell)$ (i.e., $u_i \in \mathcal{I}(\mathbb{D}, n_\ell)$) (lines 8-14). For an instance $u_i \in U'$, if $n(u_i) \not\preceq n_\ell$, u_i does not dominate the instances of the objects in $\mathcal{M}(\mathbb{D}, n_\ell)$ by Lemma 4.17. Thus, if $n(u_i) \preceq n_\ell$, the map function puts u_i into the list I. After

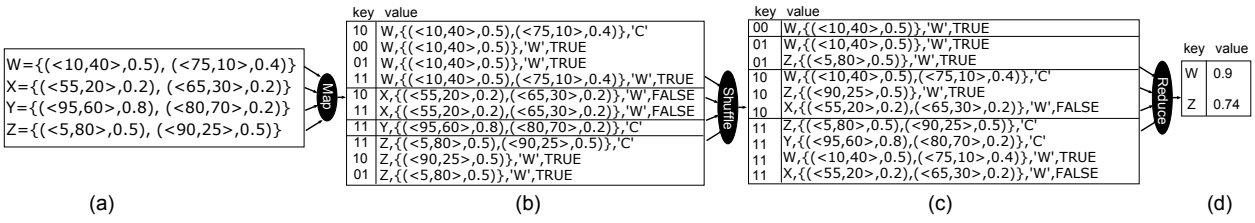


Figure 4: The steps of *PS-QPFC-MR*

every instance of U' is evaluated for n_ℓ, the map function outputs the key-value pair $\langle n_\ell, (I, \text{'W'}, cand)\rangle$ where 'W' denotes that the instances are in $\mathcal{I}(\mathbb{D}, n_\ell)$ and *cand* represents that U' is a candidate object or not (line 14). Note that when U' is a candidate object and $n_\ell = n(U'.max)$, we do nothing (line 9) since it is already sent in line 7.

Reduce function: In the shuffling phase, the key-value pairs emitted by all map functions are grouped by each distinct leaf node, and a reduce function is called with each node n_ℓ and a value list L. The value list L is split into L_C, L_W^T and L_W^F where L_C is $\mathcal{M}(\mathbb{D}, n_\ell)$, L_W^T is the subset of $\mathcal{I}_w(\mathbb{D}, n_\ell)$ whose instances are marked with $cand$ = TRUE, and L_W^F is $\mathcal{I}_w(\mathbb{D}, n_\ell) - L_W^T$ (line 1 of PS-QPFC-MR.reduce).

To split L into three partitions L_C, L_W^T and L_W^F effectively, we exploit the functionality of *secondary sorting* [19] provided by the MapReduce framework which arranges the elements in L with a specific ordering such that all elements belonging to L_C always appear first, all elements belonging to L_W^T are located next and the elements belonging to L_W^F are placed last.

Once all elements in L_C are loaded into main memory, the reduce function computes the skyline probability of every object U in L_C by invoking *SkylineProb* (lines 2-4). Since we keep only the elements of L_C in main memory, we require $O(|L_C|)$=$O(|\mathcal{M}(\mathbb{D}, n_\ell)|)$ memory.

To discover non-skyline objects earlier, we first compute $P_{sky}(U)$ with other objects in L_C since L_C is already in main memory. Then, $P_{sky}(U)$ is updated with L_W^T and next updated with L_W^F. The reason why L_W^F is read in last is that all instances in L_W^F tend to have less dominance power than the instances in L_W^T since they belong to non-skyline objects (i.e., *cand*=FALSE).

Let $\mathbb{O}$ be the set of objects whose instances were used to compute $P_{sky}(U)$ up to now. Note that, by Proposition 4.1 with $\mathbb{S} = \mathbb{O}$, the skyline probability of U computed by using $\mathbb{O}$ becomes an upper bound of $P_{sky}(U)$. Thus, whenever the skyline probability of U updated currently is less than T_p, *SkylineProb* returns zero to indicate that U is a non-skyline object. Otherwise, we output U with $P_{sky}(U)$ (lines 5-6).

EXAMPLE 5.1: *Consider the data* $\mathbb{D}$ *and the* PSQtree *in Figure* 1 *with the probability threshold* T_p=0.5. *Figures* 4(a)-(d) *show the data flow in* PS-QPFC-MR. *After the* PSQtree *is broadcast to all map functions, each map function is called with an uncertain object as illustrated in Figure* 4(a). *Consider the map function called with* X. *Since the upper bound of the skyline probability of* X *is node*(10).$P_{min} \cdot P(x_1)$ + *node*(10).$P_{min} \cdot P(x_2) = 0.4 < T_p = 0.5$, X *is not a skyline candidate object (due to Corollary 4.5). Note that every instance of* X *is contained in the region of node*(10). *The map function emits the key-value pairs* $\langle 10, (\{(\langle 55, 20\rangle, 0.2), (\langle 65, 30\rangle, 0.2)\}, \text{"W"}, False)\rangle$ *and* $\langle 11, (\{(\langle 55, 20\rangle, 0.2), (\langle 65, 30\rangle, 0.2)\}, \text{"W"}, False)\rangle$ *since node*(10) *weakly dominates node*(10) *itself and node*(11). *Figure* 4(b) *shows the key-value pairs emitted by all map functions. The key-value pairs grouped by each distinct key are provided in Figure* 4(c). *As shown in Figure* 4(d), *the probabilistic skyline objects* W *and* Z *are output by the reduce functions called with node*(10) *and node*(11), *respectively.* ∎

The continuous model: We utilize the Monte Carlo integration [9] to calculate the skyline probabilities of objects. We sample points u from $U.R$ uniformly and $P_{sky}(U)$ in Equation (2) is calculated as the average value of $|U.R| \times U.f(u) \prod_{V \in \mathbb{D}, V \neq U} P_{LS}\ (u, V)$ where $P_{LS}(u, V) = 1 - \int_{V.R} V.f(v)\mathbb{1}(v \prec u)\mathrm{d}v$. The integral to calculate $P_{LS}(u, V)$ is also computed by the Monte Carlo integration.

The pseudocode of *PS-QPFC-MR* is the same as that of *PS-QPFC-MR* for the discrete model except that it utilizes the filtering techniques for the continuous model and the lines 10-14 of the map function in Figure 3 are replaced by the lines below. Due to Lemma 4.17, when $U.R.min \prec n_\ell.max$ holds, we send U to the reduce function of n_ℓ.

```
10.  if U.R.min ≺ n_ℓ.max then
11.      emit(n_ℓ, (U, 'W', cand));
```

5.2 Optimizations of PS-QPF-MR

When a map function is invoked with an object V, each instance $v_j \in V$ is transmitted to the reduce function corresponding to every leaf node n_ℓ dominated weakly by $n(v_j)$ (i.e., the leaf node whose region includes v_j). To minimize the number of transmissions by all map functions, we can actually cluster the leaf nodes of a *PSQtree* into several groups such that a single reduce function processes all leaf nodes of each group with the main memory available in each machine.

When we cluster the leaf nodes, we should balance workloads for all reduce functions too. Let a group G_i be a set of leaf nodes $\{n_{i_1}, \ldots, n_{i_{|G_i|}}\}$. The input of a reduce function with a group G_i consists of the objects whose max points are in the region of a leaf node $n_{i_k} \in G_i$ and the instances v_j such that $n(v_j)$ weakly dominates a leaf node $n_{i_k} \in G_i$. Thus, we estimate the number of the objects as well as the number of the instances in each group by utilizing the sample used to build a *PSQtree* and force the input size of every reduce function to be similar for workload balancing.

5.2.1 Reducing Network Overhead by Clustering

Let $\mathcal{G}$ be a set of groups $\{G_1, \cdots, G_{|\mathcal{G}|}\}$ where G_i is a group of leaf nodes $\{n_{i_1}, \ldots, n_{i_{|G_i|}}\}$. Then, let $\mathcal{M}(\mathbb{D}, G_i) = \bigcup_{n_{i_k} \in G_i} \mathcal{M}(\mathbb{D}, n_{i_k})$ and $\mathcal{I}_w(\mathbb{D}, G_i) = \bigcup_{n_{i_k} \in G_i} \mathcal{I}_w(\mathbb{D}, n_{i_k})$. The reduce function called for a group G_i computes the skyline probability of every object in $\mathcal{M}(\mathbb{D}, G_i)$ by using the other objects in $\mathcal{M}(\mathbb{D}, G_i)$ and all instances in $\mathcal{I}_w(\mathbb{D}, G_i)$.

As mentioned in Section 5.1, the reduce function called for each leaf node n_ℓ requires $O(|\mathcal{M}(\mathbb{D}, n_\ell)|)$ memory only since we utilize the secondary sorting. Let the size of main memory be $s(mem)$ and the average size of an object be $s(obj)$. When we group leaf nodes, since each reduce function for a group G_i requires $O(|\mathcal{M}(\mathbb{D}, G_i)|)$ memory, we should have $|\mathcal{M}(\mathbb{D}, G_i)| \cdot s(obj) \leq s(mem)$ so that $\mathcal{M}(\mathbb{D}, G_i)$ can be kept in the main memory. In addition, since the number of transmissions by all map functions is $\sum_{G_i \in \mathcal{G}}(|\mathcal{I}_w(\mathbb{D}, G_i)| +$

$|\mathcal{M}(\mathbb{D}, G_i)|$) and $\sum_{G_i \in \mathcal{G}} |\mathcal{M}(\mathbb{D}, G_i)|$ is a constant regardless of leaf node grouping, we should minimize $\sum_{G_i \in \mathcal{G}} |\mathcal{I}_w(\mathbb{D}, G_i)|$ to reduce the number of transmissions. Therefore, our leaf node grouping problem can be formulated as follows:

DEFINITION 5.2: **[Leaf node grouping problem]** *Let the average size of an object be $s(obj)$, the size of main memory assigned to each reduce function be $s(mem)$ and $N=\{n_1, \cdots, n_{|N|}\}$ be the set of all leaf nodes in a* PSQtree. *Assume $|\mathcal{M}(\mathbb{D}, n_\ell)| \cdot s(obj) \leq s(mem)$ for every $n_\ell \in N$. The problem is to find a set of disjoint groups $\mathcal{G}=\{G_1, \cdots, G_{|\mathcal{G}|}\}$ such that $G_1 \cup \cdots \cup G_{|\mathcal{G}|} = N$, $|\mathcal{M}(\mathbb{D}, G_i)| \cdot s(obj) \leq s(mem)$ for all $i=1, \ldots, |\mathcal{G}|$ and $\sum_{G_i} |\mathcal{I}_w(\mathbb{D}, G_i)|$ is minimized.*

Since this problem can be reduced from the well-known NP-Complete *bin packing problem* [25] by setting $|\mathcal{I}_w(\mathbb{D}, G_i)| = 1$ for every group G_i, it is NP-Complete and thus we devise a greedy algorithm. Let $\tilde{\mathcal{G}}$ be the set of groups created so far in our algorithm. It takes each leaf node n_ℓ of a *PSQtree* one by one and inserts n_ℓ into the group $G_i \in \tilde{\mathcal{G}}$ which can accommodate n_ℓ (i.e., $|\mathcal{M}(\mathbb{D}, G_i \cup \{n_\ell\})| \cdot s(obj) \leq s(mem)$) with the minimum of $(|\mathcal{I}_w(\mathbb{D}, G_i \cup \{n_\ell\})| - |\mathcal{I}_w(\mathbb{D}, G_i)|)$. If there is no group to accommodate n_ℓ, we create an empty group G_j, put n_ℓ into G_j and insert G_j into $\tilde{\mathcal{G}}$.

To apply our heuristics, we need $|\mathcal{M}(\mathbb{D}, G_i)|$ and $|\mathcal{I}_w(\mathbb{D}, G_i)|$. With the sample $\mathbb{S}$ to build the *PSQtree*, by assuming that $|\mathcal{M}(\mathbb{D}, G_i)|$ and $|\mathcal{I}_w(\mathbb{D}, G_i)|$ are proportional to $|\mathcal{M}(\mathbb{S}, G_i)|$ and $|\mathcal{I}_w(\mathbb{S}, G_i)|$ respectively, we estimate them as $|\hat{\mathcal{M}}(\mathbb{D}, G_i)| = |\mathcal{M}(\mathbb{S}, G_i)| \cdot \frac{|\mathbb{D}|}{|\mathbb{S}|}$ and $|\hat{\mathcal{I}}_w(\mathbb{D}, G_i)| = |\mathcal{I}_w(\mathbb{S}, G_i)| \cdot \frac{|\mathbb{D}|}{|\mathbb{S}|}$.

5.2.2 Workload Balancing of Reduce Functions

After applying leaf node grouping, $|\mathcal{M}(\mathbb{D}, G_i)|$ of every group $G_i \in \mathcal{G}$ becomes similar and the sum of $|\mathcal{I}_w(\mathbb{D}, G_i)|$s over all groups $G_i \in \mathcal{G}$ is minimized. However, since the sizes of $\mathcal{I}_w(\mathbb{D}, G_i)$s may be skewed, the execution times of reduce functions can be quite different. Let the input of the reduce function for a group G_i be $X(G_i)$ which actually consists of $\mathcal{M}(\mathbb{D}, G_i)$ and $\mathcal{I}_w(\mathbb{D}, G_i)$. We balance the workloads of reduce functions for the groups G_i with large $|\mathcal{I}_w(\mathbb{D}, G_i)|$ by splitting $\mathcal{I}_w(\mathbb{D}, G_i)$ into m_{G_i} disjoint partitions $\{\mathcal{I}_w(\mathbb{D}, G_i, 1), \cdots, \mathcal{I}_w(\mathbb{D}, G_i, m_{G_i})\}$ such that every instance of the each object is in the same partition. With respect to $X(G_i)$, we next generate a set $\mathcal{X}(G_i)=\{X_1(G_i), \cdots, X_{m_{G_i}}(G_i)\}$ where $X_k(G_i)$ is composed of $\mathcal{M}(\mathbb{D}, G_i)$ and a partition $\mathcal{I}_w(\mathbb{D}, G_i, k)$, and invoke a reduce function with $X_k(G_i)$ to calculate partial skyline probability of each instance u of an object U in $\mathcal{M}(\mathbb{D}, G_i)$. Then, the skyline probability of U is computed in the main function by collecting all partial skyline probabilities of every instance $u \in U$.

The skyline probability of each instance u of every object $U \in \mathcal{M}(\mathbb{D}, G_i)$ can be computed by using the reduce functions each of whose input is $X_k(G_i) \in \mathcal{X}(G_i)$. Given a set of partitions $I(G_i) = \{\mathcal{I}_w(\mathbb{D}, G_i, 1), \cdots, \mathcal{I}_w(\mathbb{D}, G_i, m_{G_i})\}$ of $\mathcal{I}_w(\mathbb{D}, G_i)$, let $S(\mathbb{D}, G_i, k)$ be the set of objects whose instances are contained in the k-th partition $\mathcal{I}_w(\mathbb{D}, G_i, k) \in I(G_i)$ and $P(\mathbb{D}, G_i, k)$ be the probability that every instance v_j of an object V in $S(\mathbb{D}, G_i, k)$ which dominates u does not exist in a possible world (i.e., $P(\mathbb{D}, G_i, k) = \prod_{V \in S(\mathbb{D}, G_i, k)} (1 - \sum_{v_j \in V, v_j \prec u} P(v_j))$). For the instance u, since every object V such that there exists an instance $v_j \in V$ dominating u is contained in one of $\mathcal{M}(\mathbb{D}, G_i)$, $S(\mathbb{D}, G_i, 1)$, $\cdots$, $S(\mathbb{D}, G_i, m_{G_i}-1)$ and $S(\mathbb{D}, G_i, m_{G_i})$, the skyline probability of u can be computed as follows:

$$P_{sky}(u) = P(u) \times \prod_{V \in \mathbb{D}, V \neq U} (1 - \sum_{v_j \in V, v_j \prec u} P(v_j))$$

$$= P(u) \times \prod_{V \in \mathcal{M}(\mathbb{D}, G_i), V \neq U} (1 - \sum_{v_j \in V, v_j \prec u} P(v_j)) \times \prod_{k=1}^{m_{G_i}} P(\mathbb{D}, G_i, k)$$

While the reduce function invoked with $X_k(G_i)$ computes $P(\mathbb{D}, G_i, k)$ using $\mathcal{I}_w(\mathbb{D}, G_i, k)$, one of the reduce functions calculates $P(u) \times \prod_{V \in \mathcal{M}(\mathbb{D}, G_i), V \neq U} (1 - \sum_{v_j \in V, v_j \prec u} P(v_j))$. Then, we can compute the skyline probability of u by using the above equation.

After leaf node grouping, the number of reduce functions processed by each machine is either $\lfloor |\mathcal{G}|/t \rfloor$ or $\lfloor |\mathcal{G}|/t \rfloor + 1$ where t is the number of machines. Thus, we set the number of reduce function calls to $\lceil \frac{|\mathcal{G}|}{t} \rceil \cdot t$ which is at least $|\mathcal{G}|$ and the smallest multiple of t so that each machine processes the same number (i.e., $\lceil |\mathcal{G}|/t \rceil$) of reduce functions. To do this, our workload balancing problem is defined as follows:

DEFINITION 5.3: **[Workload balancing problem]** *Given a set of groups $\mathcal{G} = \{G_1, \cdots, G_{|\mathcal{G}|}\}$ which is the result of the leaf node grouping problem and a number of machines t, the problem is to find $\mathcal{X}(G_i) = \{X_1(G_i), \cdots, X_{m_{G_i}}(G_i)\}$ such that (1) $\mathcal{I}_w(\mathbb{D}, G_i)$ is split into disjoint partitions $I(G_i) = \{\mathcal{I}_w(\mathbb{D}, G_i, 1), \cdots, \mathcal{I}_w(\mathbb{D}, G_i, m_{G_i})\}$ for each group $G_i \in \mathcal{G}$, (2) $X_k(G_i)$ is composed of $\mathcal{M}(\mathbb{D}, G_i)$ as well as $\mathcal{I}_w(\mathbb{D}, G_i, k) \in I(G_i)$, (3) $\lceil \frac{|\mathcal{G}|}{t} \rceil \cdot t = \sum_{G_i} m_{G_i}$ and (4) $\max_{G_i \in \mathcal{G}, X_k(G_i) \in \mathcal{X}(G_i)} |\mathcal{I}_w(\mathbb{D}, G_i, k)|$ is minimized. Note that $\sum_{G_i} m_{G_i}$ is the total number of reduce functions utilized by all groups.*

We next present the greedy algorithm *GreedyWorkload* for the workload balancing problem. Let $\tilde{m}_{G_i}$ be the number of partitions in $\mathcal{X}(G_i)$ for each group G_i. Initially, $\tilde{m}_{G_i}=1$. At each step of *GreedyWorkload*, we repeatedly select the group G_i with the maximum $|\mathcal{I}_w(\mathbb{D}, G_i)|/\tilde{m}_{G_i}$ and increase $\tilde{m}_{G_i}$ by one until $\sum_{G_i} \tilde{m}_{G_i} = \lceil \frac{|\mathcal{G}|}{t} \rceil \cdot t$. As we did in leaf node grouping previously, we estimate $|\mathcal{I}_w(\mathbb{D}, G_i)|$ by utilizing a sample $\mathbb{S}$ of the objects in $\mathbb{D}$.

After *GreedyWorkload* terminates, for every group G_i, we split $\mathcal{I}_w(\mathbb{D}, G_i)$ into $\{\mathcal{I}_w(\mathbb{D}, G_i, 1), \cdots, \mathcal{I}_w(\mathbb{D}, G_i, \tilde{m}_{G_i})\}$. We broadcast $\mathcal{G}$ and $\tilde{m}_{G_i}$ of every $G_i \in \mathcal{G}$ to all map functions. To make the size of every partition similar, when a map function is called with an object whose instances belong to $\mathcal{I}_w(\mathbb{D}, G_i)$, the map function chooses a random number k between 1 and $\tilde{m}_{G_i}$ and sends the instances to the reduce function handling $X_k(G_i)$.

LEMMA 5.4: *When $\mathcal{I}_w(\mathbb{D}, G_i)$ can be split into equi-sized partitions for every $G_i \in \mathcal{G}$, the procedure* GreedyWorkload *finds an optimal solution for the workload balancing problem.*

Proof: Due to the space limitation, we omit the proof. ∎

Since we may not split $\mathcal{I}_w(\mathbb{D}, G_i)$ into equi-sized partitions such that every instance of each object lies in the same partition, *GreedyWorkload* does not guarantee the optimality.

5.3 Sample Size and Split Threshold of a PSQtree

In leaf node grouping, although we require $s(obj) \cdot |\mathcal{M}(\mathbb{D}, G_i)| \leq s(mem)$ for every group G_i, the reduce function handling G_i may suffer from the lack of memory space since we estimate $|\mathcal{M}(\mathbb{D}, G_i)|$ approximately by using a sample $\mathbb{S}$ of $\mathbb{D}$. To avoid such deficiency, we enforce $s(obj) \cdot |\hat{\mathcal{M}}(\mathbb{D}, G_i)| \leq \alpha \cdot s(mem)$ (e.g., $\alpha=0.8$) where $|\hat{\mathcal{M}}(\mathbb{D}, G_i)|$ is the estimate of $|\mathcal{M}(\mathbb{D}, G_i)|$. We refer to it as the *memory utilization heuristics*.

Finding a proper sample size: We study how to choose the sample size to estimate $|\mathcal{M}(\mathbb{D}, G_i)|$ accurately. When $s(obj) \cdot |\hat{\mathcal{M}}(\mathbb{D}, G_i)| < \alpha \cdot s(mem)$ but $s(mem) < s(obj) \cdot |\mathcal{M}(\mathbb{D}, G_i)|$, it is problematic. Thus, we want the probability that $|\hat{\mathcal{M}}(\mathbb{D}, G_i)| < \alpha \cdot |\mathcal{M}(\mathbb{D}, G_i)|$ is less than a threshold δ.

LEMMA 5.5: *Given a group G_i, a threshold δ and a sample $\mathbb{S} \subset \mathbb{D}$, if $|\mathbb{S}| \geq \frac{-2\cdot|\mathbb{D}|\cdot\ln\delta}{(1-\alpha)^2\cdot|\mathcal{M}(\mathbb{D},G_i)|}$, we have $P[|\hat{\mathcal{M}}(\mathbb{D},G_i)| < \alpha\cdot|\mathcal{M}(\mathbb{D},G_i)|] < \delta$.*

Proof: Let X_j be a random variable that is 1 if j-th object in $\mathbb{S}$ belongs to $\mathcal{M}(\mathbb{D},G_i)$ and 0 otherwise. Since we do uniform random sampling, $X_1,\cdots,X_{|\mathbb{S}|}$ are independent Bernoulli trials with $P(X_j{=}1) = |\mathcal{M}(\mathbb{D},G_i)|/|\mathbb{D}|$. The number of objects in $\mathbb{S}$ belonging to $\mathcal{M}(\mathbb{D},G_i)$ is $X = \sum_j X_j$ and the expected value of X is $\mu{=}E[X]{=}|\mathbb{S}|\cdot|\mathcal{M}(\mathbb{D},G_i)|/|\mathbb{D}|$. We have $P[|\hat{\mathcal{M}}(\mathbb{D},G_i)| < \alpha\cdot|\mathcal{M}(\mathbb{D},G_i)|] = P[X\cdot|\mathbb{D}|/|\mathbb{S}| < \alpha\cdot|\mathcal{M}(\mathbb{D},G_i)|]$ since $|\hat{\mathcal{M}}(\mathbb{D},G_i)|$ is $X\cdot|\mathbb{D}|/|\mathbb{S}|$.

Chernoff bounds state that for $0{<}\epsilon{\leq}1$, we have $P[X{<}(1\text{-}\epsilon)\mu]{<}\exp(\text{-}\mu\epsilon^2/2)$. Rewriting the probability to conform to the Chernoff bounds, we get $P[X{<}(1\text{-}(1\text{-}\frac{\alpha\cdot|\mathbb{S}|\cdot|\mathcal{M}(\mathbb{D},G_i)|}{|\mathbb{D}|\mu}))\mu]{<}\delta$. Then, we obtain $\exp(\text{-}\frac{\mu}{2}(1\text{-}\frac{\alpha\cdot|\mathbb{S}|\cdot|\mathcal{M}(\mathbb{D},G_i)|}{|\mathbb{D}|\mu})^2){\leq}\delta$ by applying the Chernoff bounds. Substituting $\mu{=}|\mathbb{S}|\cdot|\mathcal{M}(\mathbb{D},G_i)|/|\mathbb{D}|$ and solving it for $|\mathbb{S}|$, we obtain the lower bound of $|\mathbb{S}|$. ∎

To compute the above bound for every problematic group G_i satisfying $s(obj)\cdot|\mathcal{M}(\mathbb{D},G_i)| > s(mem)$, by letting $|\mathcal{M}(\mathbb{D},G_i)| = \frac{s(mem)}{s(obj)}$, we have $|\mathbb{S}|{\geq}\frac{-2\cdot|\mathbb{D}|\cdot\ln\delta\cdot s(obj)}{(1-\alpha)^2\cdot s(mem)}$ since the lower bound of $|\mathbb{S}|$ is maximized when $|\mathcal{M}(\mathbb{D},G_i)|$ is minimized.

Setting the split threshold ρ: When we build a *PSQtree* with a sample $\mathbb{S}$, we split a node n if the number of instances in n exceeds the split threshold ρ. To apply leaf node grouping with the *memory utilization heuristics*, we should guarantee that $s(obj)\cdot|\mathcal{M}(\mathbb{D},n_\ell)|{\leq}\alpha\cdot s(mem)$ for each n_ℓ since every group G_i contains at least a single leaf node.

After the *PSQtree* is generated, we assume that the number of instances of objects appearing in each leaf node n_ℓ is at most $\rho\cdot|\mathbb{D}|/|\mathbb{S}|$. Let n_I be the average number of instances in each object. Then, under the assumption of uniform distribution, we have $|\mathcal{M}(\mathbb{D},n_\ell)| \leq \rho/n_I\cdot|\mathbb{D}|/|\mathbb{S}|$. Thus, we set $\rho = \alpha\cdot s(mem)\cdot n_I\cdot|\mathbb{S}|/(s(obj)\cdot|\mathbb{D}|)$ obtained by finding the minimum ρ satisfying $s(obj)\cdot\rho/n_I\cdot|\mathbb{D}|/|\mathbb{S}| \leq \alpha\cdot s(mem)$.

6. MAPREDUCE ALGORITHMS WITH RANDOM PARTITIONING

In this section, we present the MapReduce algorithm *PS-BRF-MR* which utilizes random partitioning as well as the filtering techniques in Section 4.1. We refer to the brute-force algorithm based on random partitioning without such filtering techniques as *PS-BR-MR*. Due to the space limitation, we omit the detailed pseudocodes for both algorithms.

Generally, random partitioning is not suitable to the continuous model since all objects required to compute the skyline probability of an object U by performing the integration in Equation (2) cannot be in the same partition containing U. To apply random partitioning to the continuous model, we adapt a specific integration method, Monte Carlo integration [9], which is based on sample points (refer to [9] for details). Thus, for each object U, the partial values required to compute the integration for the skyline probabilities are computed using the sample points selected in each partition. Then, we calculate the skyline probability of U by integrating the partial values of all partitions.

PS-BRF-MR: When a dataset $\mathbb{D}$ is split into disjoint partitions, $P_1,\ldots,P_m$, to calculate the skyline probability of an instance $u_i \in U$, we compute its *k-th local* skyline probability $P_{LS}(u_i,U,k)$ in every partition P_k.

DEFINITION 6.1: *For disjoint partitions $P_1,\ldots,P_m$ of a dataset $\mathbb{D}$ and an instance $u_i \in U$, the k-th* local skyline probability *of u_i, denoted by $P_{LS}(u_i,U,k)$, is $\prod_{V\in P_k,V\neq U}(1\text{-}\sum_{v_j\in V,v_j\prec u_i}P(v_j))$. By Equation (1), we obtain*

$$P_{sky}(u_i) = P(u_i)\prod_{k=1}^{m}P_{LS}(u_i,U,k). \quad (4)$$

The algorithm *PS-BRF-MR* consists of two MapReduce phases. In the first MapReduce phase, the filtering techniques presented in Section 4 are applied to identify the non-skyline objects so that we can compute the skyline probabilities for the skyline *candidate* objects only. In the second MapReduce phase, we gather every local skyline probability of each instance to compute the skyline probabilities of all objects. *PS-BRF-MR* consists of the following three phases:

(1) PSQtree building phase: We build a *PSQtree* with a sample $\mathbb{S} \subset \mathbb{D}$ by calling the procedure *GenQtree* in Section 4.2.2. Recall that it is done without using MapReduce.

(2) Local skyline probability phase: After broadcasting a *PSQtree* and T_p, each map function checks if each object is a *candidate* by the filtering methods in Section 4.

We divide the data objects $\mathbb{D}$ into disjoint partitions, $P_1,\ldots,P_m$. For every partition-pair (P_i,P_j) with $1{\leq}i{\leq}j{\leq}m$, we compute the local skyline probabilities of the instances in P_i and P_j in parallel. For each partition-pair (P_i,P_j), when $i=j$, for every instance u of each candidate object U in P_i, we compute the i-th local skyline probability $P_{LS}(u,U,i)$ defined in Definition 6.1. When $i<j$, we compute $P_{LS}(u,U,j)$ for every u of U in P_i by considering the instances $v\in V$ in P_j and calculate $P_{LS}(v,V,i)$ for every v of each candidate object V in P_j by considering the instances u in P_i. To reduce the number of comparisons, we compare the skyline candidate objects with other skyline candidate objects first and then compare them to non-skyline objects by using the secondary sorting illustrated in Section 5.1.

(3) Global skyline phase: We gather the local skyline probabilities computed in the previous phase and calculate the exact skyline probabilities of the instances of every skyline candidate object using Equation (4). For a candidate object U, if $P_{sky}(U) = \sum_{u\in U}P(u)\prod_{i=1}^{m}P_{LS}(u,U,i) \geq T_p$, we output U as a skyline object.

The continuous model: For the continuous model, we use a specific integration method, Monte Carlo integral [9] which samples points u from the uncertainty region $U.R$ uniformly. In the local skyline probability phase, for each partition-pair (P_i,P_j), when $P_i{=}P_j$, it calculates $\prod_{V\in P_i,U\neq V}P_{LS}(u,V)$ for all $U\in P_i$ where $P_{LS}(u,V)$ is $1-\int_{V.R}V.f(v)\mathbb{1}(v\prec u)\mathrm{d}v$. If $P_i{\neq}P_j$, we compute $\prod_{V\in P_j}P_{LS}(u,V)$ for $U\in P_i$ and $\prod_{U\in P_i}P_{LS}(v,U)$ for $V\in P_j$. In the global skyline phase, we compute $P_{sky}(U)$ by utilizing the $\prod_{V\in P_i,U\neq V}P_{LS}(u,V)$ obtained in the previous phase since $P_{sky}(U)$ is the average value of $|U.R|\times U.f(u)\prod_{V\in\mathbb{D},V\neq U}P_{LS}(u,V)$ by using Monte Carlo integration as in Section 5.1.

Algorithm	Description
PS-QP-MR	The algorithm with quadtree partitioning
PS-QPF-MR	The algorithm with quadtree partitioning and filtering
PS-BR-MR	The algorithm with random partitioning
PS-BRF-MR	The algorithm with random partitioning and filtering
PSMR	The state-of-the-art algorithm in [14]

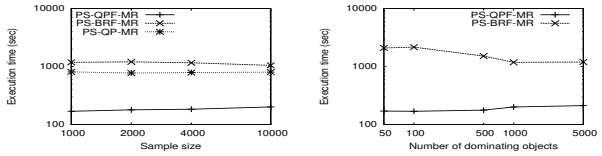
Table 1: Implemented algorithms

7. EXPERIMENTS

Experiments were done mainly on a cluster of 50 machines with Intel i3 3.3GHz CPU and 4GB of memory running

Parameter	Range	Default
No. of samples ($\|\mathbb{S}\|$)	1000 ∼ 10,000	1000 for *PS-QPF-MR* 2000 for *PS-QP-MR* 10000 for *PS-BRF-MR*
No. of dominating objects ($\|F\|$)	50 ∼ 5000	100 for *PS-QPF-MR* 1000 for *PS-BRF-MR*
No. of objects ($\|\mathbb{D}\|$)	$10^5 \sim 10^8$	10^7
No. of dimensions (d)	2 ∼ 8	4
Probability threshold (T_p)	0.1 ∼ 0.6	0.3
No. of inst. per object (ℓ)	1 ∼ 400	40
No. of machines (t)	10 ∼ 200	25

Table 2: Parameters



(a) Varying $|\mathbb{S}|$ (b) Varying $|F|$

Figure 5: Selection of $|\mathbb{S}|$ and $|F|$

Linux. We also used Amazons EC2 Infrastructure as a Service (IaaS) cloud to show the scalability of *PS-QPF-MR* up to 200 machines with Intel Xeon 2.5GHz CPU and 3.75GB of memory. The implementations of all algorithms were compiled by Javac 1.6. We used Hadoop 1.2.1 for MapReduce. The execution times in the graphs are plotted in log scale. We do not report the execution times which exceed 6 hours. The tested algorithms are summarized in Table 1.

7.1 Experimental Environments

Datasets: We generated the synthetic datasets with correlated, independent and anti-correlated distributions, referred to as *COR*, *IND* and *ANTI* respectively, since they are typically used to evaluate the performance of skyline algorithms [8, 21, 22]. For a d-dimensional space, we generated the center c of each object using the three distributions where each dimension has a domain of $[1, 10000)$. In the discrete model, for each object U, we selected the number of U's instances using the uniform distribution in the range $[1, \ell]$, where ℓ is 40 by default. Each instance was generated inside the rectangle centered at c whose edge size is uniformly distributed in the range $[1, 200]$. The ratio of the objects U with $\sum_{u_i \in U} P(u_i) = 1$ to all objects in the dataset was set to 0.5. In the continuous model, for each object U, we selected the length of k-th dimension of $U.R$ in $[1, 200]$, and assumed $U.f(\cdot)$ is the uniform distribution. The sizes of resulting synthetic datasets are varied from 88MB to 86GB depending on the number of points ($|\mathbb{D}|$), the number of dimensions (d) and the number of instances per each object (ℓ). We also varied the probability threshold T_p from 0.1 to 0.6 to produce diverse probabilistic skyline queries. We set $T_p = 0.3$ as the default value. The parameters used by our algorithms are summarized in Table 2.

Default value of m: In the random partitioning algorithms (i.e., *PS-BR-MR* and *PS-BRF-MR*), we split data $\mathbb{D}$ into m partitions. Since such algorithms split all pairs of objects into $m(m+1)/2$ partition-pairs, we set m to the minimum natural number satisfying $m(m+1)/2 \geq t$ so that each machine can process at least a single partition-pair.

Default values of $|\mathbb{S}|$ and $|F|$: By the discussion in Section 5.3, the sample size $|\mathbb{S}|$ should be at least 700 objects since $|\mathbb{S}| \geq \frac{-2 \cdot |\mathbb{D}| \cdot \ln \delta \cdot s(obj)}{(1-\alpha)^2 \cdot s(mem)}$=700 with $s(mem)$=4GB, $s(obj)$= 1KB, $|\mathbb{D}|$=10^7, α=0.8 and δ=0.01. Thus, to find the proper sizes of a sample $\mathbb{S}$ and a dominating object set F (i.e., $|\mathbb{S}|$ and $|F|$), we ran our algorithms with varying $|\mathbb{S}|$ from $1,000$ to $10,000$ and $|F|$ from 50 to $5,000$, respectively. The average execution times over all datasets with varying $|\mathbb{S}|$ and $|F|$ are shown in Figures 5(a) and 5(b), respectively. Since *PS-BR-MR* does not utilize a *PSQtree* and a dominating object set F, we do not plot its execution times in Figure 5.

T_p	*PS-QPF -MR* (sec)	*PS-BRF -MR* (sec)	# of candidate objects	# of skyline objects
0.1	400	1905	259009	1057
0.2	223	1469	204964	509
0.3	164	1452	165907	329
0.4	161	1267	140129	234
0.5	151	1164	121678	172
0.6	151	1115	106530	127

Table 3: Varying the probability threshold (T_p)

Although more objects are filtered by the upper-bound and dominance-power filtering as $|\mathbb{S}|$ and $|F|$ increase, the costs for computing upper bounds and maintaining dominating object set increase. Consequently, we set the default values of $|\mathbb{S}|$ and $|F|$ with which each algorithm show the best performance. For instance, the best performance of *PS-QPF-MR* is obtained with $|\mathbb{S}| = 1000$ and $|F| = 100$.

7.2 Performance Analysis

We presented the experiment results with the discrete model first and the continuous model next.

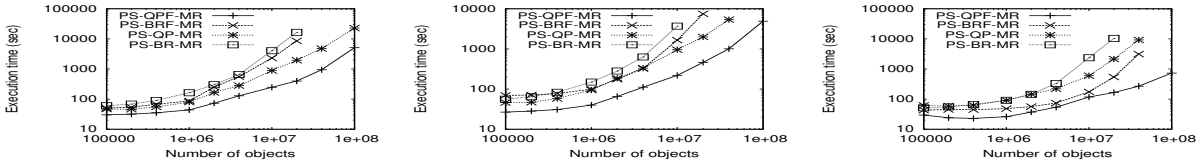
Varying $|\mathbb{D}|$: We plotted the running times of the tested algorithms with varying the number of objects $|\mathbb{D}|$ from 10^5 to 10^8 with each dataset in Figures 6(a), (b) and (c), respectively. *PS-QPF-MR* with *COR* is faster than that with the other datasets since most of instances are dominated by a few instances in *COR* and the three filtering methods can identify non-skyline objects effectively. The best performance is shown by *PS-QPF-MR* which utilizes the three filtering methods and the quadtree partitioning. *PS-QPF-MR* is also found to be at least 1.7 times faster than *PS-BRF-MR*. Since *PS-QPF-MR* and *PS-BRF-MR* are always faster than *PS-QP-MR* and *PS-BR-MR*, respectively, due to the three filtering methods, we showed only the execution times of *PS-QPF-MR* and *PS-BRF-MR* in the rest of the paper.

Varying d: The execution times with varying the number of dimensions d from 2 to 8 were reported in Figure 7. Since the time complexity of checking the dominance relationship between instances is $O(d)$, the execution times of both algorithms become larger as d grows. We found that *PS-QPF-MR* is 4.4 times faster than *PS-BRF-MR* on the average since quadtree partitioning is very effective. However, *PS-BRF-MR* performs fast for *COR* with high dimension since there are a small number of candidate objects and merging their skyline probabilities can be done quickly.

Varying T_p: We showed the execution time, number of candidate objects and number of skyline objects on average with varying T_p from 0.1 to 0.6 in Table 3. Since all filtering methods are applied before data partitioning, the average numbers of candidate objects by both algorithms are the same. With increasing T_p, since the numbers of candidate and skyline objects decrease, the execution times decrease.

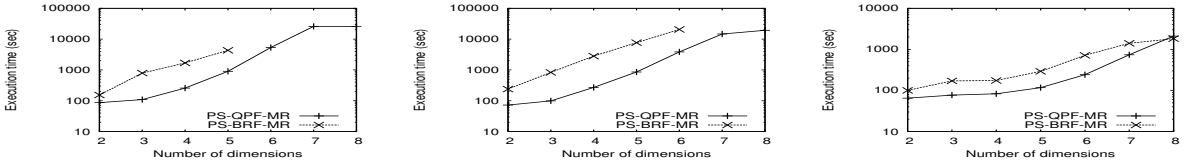
	t	10	20	25	30	40	50
PS-QPF -MR ($\|\mathbb{D}\| = 10^7$)	*IND*	401	242	212	197	167	162
	COR	177	89	85	79	78	64
	ANTI	429	228	196	175	152	135
PS-BRF -MR ($\|\mathbb{D}\| = 10^7$)	*IND*	4373	2089	1872	1469	1177	912
	COR	361	205	179	160	130	117
	ANTI	4893	2409	2307	1664	1338	959
PS-QPF -MR ($\|\mathbb{D}\| = 10^8$)	*IND*	8107	4555	3569	2698	2268	1811
	COR	1119	627	541	471	398	351
	ANTI	8442	3874	3738	3002	2206	1987

Table 4: Varying t with our cluster (sec)



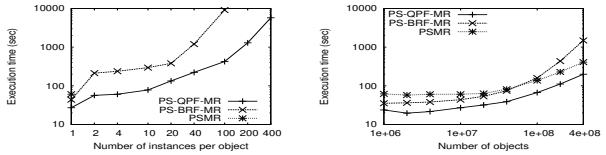
(a) ANTI (b) IND (c) COR

Figure 6: Varying the number of objects ($|\mathbb{D}|$)



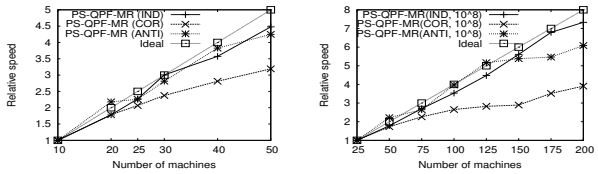
(a) ANTI (b) IND (c) COR

Figure 7: Varying the number of dimensions (d)



(a) Varying ℓ (b) Varying $|\mathbb{D}|$ when $\ell = 1$

Figure 8: Varying ℓ and $|\mathbb{D}|$ when $\ell = 1$



(a) With our cluster (b) With Amazon EC2

Figure 9: Relative speedups with $|\mathbb{D}| = 10^8$

PS-QPF-MR is up to 7.9 times faster than *PS-BRF-MR*.

Varying ℓ: We evaluated both algorithms with changing the number of instances per object ℓ from 1 to 400. We also tested the state-of-the-art algorithm *PSMR* for the specific case where each object has only a single instance. We showed the average execution times over all datasets in Figure 8(a). Since *PSMR* is only applicable when ℓ=1, we plotted the execution time of *PSMR* only when ℓ=1. Although *PS-BRF-MR* is worse than *PSMR* with large datasets, *PS-QPF-MR* is 2.1 times faster than *PSMR*. We also reported the execution times of all algorithms with varying $|\mathbb{D}|$ from 10^5 to 4×10^8 when ℓ=1 in Figure 8(b). We found that *PS-QPF-MR* is 2.3 times faster than *PSMR* on the average.

Varying t: With increasing the number of machines t up to 50 in our cluster, we presented the execution times with the default-sized datasets ($|\mathbb{D}|$=10^7) and large datasets ($|\mathbb{D}|$=10^8) in Table 4. For the large datasets, since *PS-BRF-MR* finished within 6 hours only when $t = 40, 50$ with COR, we reported execution times and *relative speedup* to 10 machines of *PS-QPF-MR* only in Table 4 and Figure 9(a), respectively.

To show the scalability of *PS-QPF-MR*, we also tested with large datasets ($|\mathbb{D}|$=10^8) on Amazon EC2 Infrastructure consisting of 200 machines and showed the execution time as well as *relative speedup* to 25 machines in Table 5 and Figure 9(b), respectively.

In both experiments using large datasets, *PS-QPF-MR* shows linear speedup with *IND* and *ANTI*, but sub-linear

t	25	50	75	100	125	150	175	200
IND	8783	4936	3252	2485	1961	1565	1293	1198
COR	1234	712	546	466	437	426	351	316
ANTI	12655	5713	4783	3186	2451	2352	2315	2080

Table 5: Varying t on Amazon EC2 with $|\mathbb{D}|$=10^8 (sec)

Filtering technique	IND	COR	ANTI
Zero-probability (# of inst.)	12806654	172962353	4352818
Upper-bound (# of obj.)	882787	8614581	490691
Dominance-power (# of obj.)	9773641	9986907	9605070

Table 6: Filtered objects per filtering technique

Dataset	*IND*	*COR*	*ANTI*	Average
PS-QPF-MR (ALL)	212	85	196	164
PS-QPF-MR (D)	226	123	207	185

Table 7: Effects of the filtering techniques (sec)

speedup with *COR*. It is because the number of probabilistic skyline objects in the correlated data is very small and the benefit of using a large number of machines is marginal.

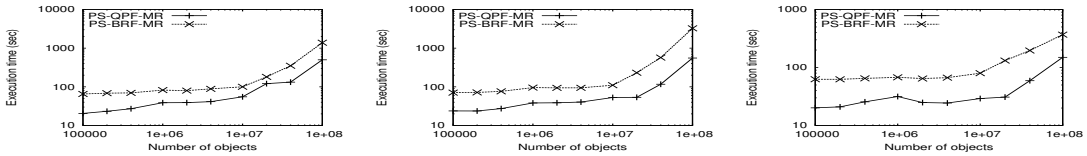
The effects of filtering techniques: We first presented the number of instances removed for zero-filtering technique and the numbers of non-skyline objects detected not to compute their skyline probabilities for each of the other filtering techniques in Table 6. We found that dominance-power filtering detects more non-skyline objects than upper-bound filtering. In Table 7, we showed the execution times of *PS-QPF-MR* by applying dominance-power filtering only (D) or all filtering techniques (ALL). When all filtering techniques were used, we applied them in the order of zero-probability filtering, upper-bound filtering and dominance-power filtering. Applying all filtering techniques is faster than applying dominance-power filtering only in *PS-QPF-MR*.

The effects of optimization techniques: In Table 8, we reported the average execution times and average number of transmitted instances by *PS-QPF-MR* without leaf node grouping and workload balancing (NONE), *PS-QPF-MR* with leaf node grouping only (L) and *PS-QPF-MR* with both methods (L and W). *PS-QPF-MR* with leaf node grouping (L) has 49% of transmitted instances than *PS-QPF-MR* without both methods (NONE). Since the workload balancing technique splits the instances required to compute the skyline probabilities of objects in each group in order to utilize all machines available, *PS-QPF-MR* with L and W is the most efficient as shown in Table 8.

The effect of quadtree partitioning: To show the effectiveness of quadtree partitioning, we experimented with datasets of $|\mathbb{D}| = 10^7$ using 200 machines on Amazon EC2 and presented the execution times as well as the numbers of checking dominance relationships between instances of objects by both algorithms in Table 9. While *PS-QPF-MR* has 1.37 times smaller number of dominance relationship comparisons than *PS-BRF-MR*, *PS-QPF-MR* is 2.33 times

PS-QPF-MR	L and W	L	NONE
Execution time (sec)	164	301	329
# of transmitted instances ($\times 10^6$)	454	454	894

Table 8: Effects of optimization techniques



(a) ANTI (b) IND (c) COR

Figure 10: Varying the number of objects ($|\mathbb{D}|$) for the continuous model

$\|\mathbb{D}\| = 10^7$	Algorithm	IND	COR	ANTI
Execution time (sec)	*PS-QPF-MR*	164	82	198
	PS-BRF-MR	473	143	470
# of dominance comparisons ($\times 10^6$)	*PS-QPF-MR*	59579	2037	58834
	PS-BRF-MR	72844	3432	70369

Table 9: Effect of quadtree partitioning using EC2

faster than *PS-QPF-MR*, on average. Since *PS-QPF-MR* has a single MapReduce phase but *PS-BRF-MR* consists of two MapReduce phases, the performance gain in terms of execution time is higher than that in terms of dominance relationship comparisons for *PS-QPF-MR*.

The continuous model: We set the default values of $(|\mathbb{S}|, |F|)$ to $(10000, 2000)$ and $(2000, 1000)$ for *PS-QPF-MR* and *PS-BRF-MR*, respectively, since they performed the best with those values. We omit the experimental results with varying $|\mathbb{S}|$ and $|F|$ since they show similar patterns with those for the discrete model. In Figure 10, we plotted the execution times of both algorithms with varying $|\mathbb{D}|$. We found that *PS-QPF-MR* runs up to 7.72 times faster and is 2.37 times faster on the average than *PS-BRF-MR*.

8. CONCLUSION

We studied the probabilistic skyline query processing on MapReduce for both discrete and continuous models. To identify probabilistic non-skyline objects in advance, we proposed the upper-bound, zero-probability and dominance-power filtering techniques. To get the probabilistic skyline with a single MapReduce phase, we developed the algorithm *PS-QP-MR* by using a *PSQtree* to distribute the instances of objects effectively. We next devised the algorithm *PS-QPF-MR* by additionally applying the three filtering techniques to *PS-QP-MR*. We also proposed the leaf node grouping technique to reduce network overhead and the workload balancing technique to balance the workload of reduce functions based on the number of machines available. As baseline algorithms, we developed the algorithm *PS-BR-MR* with two MapReduce phases by using random partitioning only and the algorithm *PS-BRF-MR* by applying the three filtering techniques to *PS-BR-MR*. Our experiments showed that *PS-QPF-MR* is much faster and more scalable than the other algorithms. As future work, we will study the parallel algorithms for probabilistic dynamic and reverse skyline queries.

Acknowledgment

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Plannig (NRF-2012M3C4A7033342). This research was also supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2015-H8501-15-1013) supervised by the IITP(Institute for Information & communication Technology Promotion) as well as supported by Defense Acquisition Program Administration and Agency for Defense Development under the contract UD140022PD, Korea.

9. REFERENCES

[1] E. Ada and C. Re. Managing uncertainty in social networks. *IEEE Data Eng. Bull.*, 30(2):15–22, 2007.
[2] F. N. Afrati, P. Koutris, D. Suciu, and J. D. Ullman. Parallel skyline queries. In *ICDT*, pages 274–284, 2012.
[3] L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and simple relational processing of uncertain data. In *ICDE*, 2008.
[4] Apache. Apache hadoop. http://hadoop.apache.org, 2010.
[5] M. J. Atallah and Y. Qi. Computing all skyline probabilities for uncertain data. In *PODS*, 2009.
[6] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, 1990.
[7] C. Böhm, F. Fiedler, A. Oswald, C. Plant, and B. Wackersreuther. Probabilistic skyline queries. In *CIKM*, pages 651–660. ACM, 2009.
[8] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.
[9] R. E. Caflisch. Monte carlo and quasi-monte carlo methods. *Acta numerica*, 7:1–49, 1998.
[10] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 523–544, 2007.
[11] J. Dean and S. Ghemawat. Mapreduce: a flexible data processing tool. *Communications of the ACM*, 53(1), 2010.
[12] E. Dellis and B. Seeger. Efficient computation of reverse skyline queries. In *VLDB*, pages 291–302, 2007.
[13] A. Deshpande, C. Guestrin, and S. Madden. Using probabilistic models for data management in acquisitional environments. In *CIDR*, pages 317–328, 2005.
[14] L. Ding, G. Wang, J. Xin, and Y. Yuan. Efficient probabilistic skyline query processing in mapreduce. In *BigData*, pages 203–210, 2013.
[15] X. L. Dong, A. Halevy, and C. Yu. Data integration with uncertainty. In *VLDB*, pages 469–500, 2009.
[16] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB*, pages 275–286, 2002.
[17] X. Lian and L. Chen. Monochromatic and bichromatic reverse skyline search over uncertain databases. In *SIGMOD*, pages 213–226, 2008.
[18] V. Ljosa and A. K. Singh. Top-k spatial joins of probabilistic objects. In *ICDE*, pages 566–575, 2008.
[19] A. Metwally and C. Faloutsos. V-smart-join: A scalable mapreduce framework for all-pair similarity joins of multisets and vectors. *PVLDB*, 5(8):704–715, 2012.
[20] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD*, 2003.
[21] Y. Park, J.-K. Min, and K. Shim. Parallel computation of skyline and reverse skyline queries using mapreduce. In *VLDB*, 2013.
[22] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *VLDB*, pages 15–26, 2007.
[23] M. A. Soliman, I. F. Ilyas, and K.-C. Chang. Top-k query processing in uncertain databases. In *ICDE*, 2007.
[24] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *VLDB*, pages 301–310, 2001.
[25] V. V. Vazirani. *Approximation algorithms*. springer, 2001.
[26] Y. Wang, X. Li, X. Li, and Y. Wang. A survey of queries over uncertain data. *Knowledge and information systems*, 37(3):485–530, 2013.
[27] B. Zhang, S. Zhou, and J. Guan. Adapting skyline computation to the mapreduce framework: Algorithms and experiments. In *DASFAA*, pages 403–414, 2011.