ProbSky: Efficient Computation of Probabilistic Skyline Queries Over Distributed Data

Ai-Te Kuo, Haiquan Chen, Liang Tang,
Wei-Shinn Ku, *Senior Member, IEEE*, and Xiao Qin, *Senior Member, IEEE*

Abstract—Skyline queries have been widely used in various application domains including multi-criteria decision making, search pruning, and personalized recommendation systems. Given multiple criteria, skyline queries prune the search space of a large collection of multi-dimensional objects to a small set by returning objects that are not dominated by or superior to others. As an extension of the traditional skyline queries, probabilistic skyline queries aim to cope with uncertain datasets. This paper presents a novel MapReduce-based framework, ProbSky, in support of fast parallel distributed evaluation of probabilistic skyline queries on large high-dimensional data. ProbSky is adept at efficiently evaluating exact p-skyline queries on large uncertain data without compromising the quality of query results. From the theoretical point of view, we formally prove two pruning lemmas integrated with ProbSky to strengthen the early pruning capacity. ProbSky builds on top of three optimization techniques: dominant instance pruning, slab-based partitioning, and reference point-based acceleration. These extensive experiments driven by both real and synthetic datasets, reveal that compared to the state-of-the-art methods ProbSky speeds up the evaluation of the exact p-skyline queries on large high-dimensional data by at least one order of magnitude in most cases. Our experimental results also validate that by balancing the memory consumption and execution time among machines, ProbSky is adroit at curbing the bottleneck effect that causes severe system performance deterioration.

Index Terms—Decision making, distributed databases, parallel programming, distributed programming, MapReduce, skyline query, probabilistic skyline query, query processing

1 Introduction

Skyline queries [1], [2], [3], [4] have been widely used in a diversity of application domains such as multi-criteria decision analysis, search space pruning, and personalized recommendation systems. Given more than one criterion, the traditional skyline queries return points that are not dominated by or superior to others. Due to the surge of data with uncertainties, a vast amount of literature has laid emphasis on the uncertain database management in fields of study, such as probabilistic ranking query [5], probabilistic k-NN query [6], and probabilistic top-k query [7]. The probabilistic skyline queries were proposed to cope with uncertain datasets [8] as an extension of the traditional skyline queries. Unlike the traditional skyline queries, the probabilistic skyline queries represent an object by a probability density function (PDF) rather than a single representative value, and thus provide more accurate and informative skyline analysis on uncertain data. Given a probability threshold p, the probabilistic skyline queries find the exact skyline probabilities of the uncertain objects and return a set of uncertain objects whose skyline probabilities are not less than p. As computing the exact skyline probabilities of the uncertain objects are computationally expensive, a few solutions [8], [9], [10], [11] have been proposed to curtail the cost.

In response to the recent highlighted interest in large-scale data processing within a reasonable time, parallel and distributed evaluation on the skyline queries has drawn extensive interest over the past decade [12], [13], [14], [15]. On the other hand, as an easy-to-use parallel and distributed computing paradigm with high scalability, reliability, and failure tolerance, MapReduce [16] has been incorporated into various data processing systems [17], [18], [19], [20]. One prominent advantage of the MapReduce model is that the developer only needs to write the Map and Reduce functions without much concern about details of parallelism across machines. However, few studies have been rolled out to take full advantage of MapReduce to make the probabilistic skyline queries scale up to high-volume data. As the probabilistic skyline queries are indecomposable, the efficient and scalable computation of the exact p-skyline queries using MapReduce presents specific challenging questions: (1) how to prune unnecessary dominance tests in an early phase to reduce the computational cost for high-volume data? and (2) how to maximize parallelism by workload balancing?

A Motivating Application. The exact p-skyline queries on uncertain data can be applied in numerous application

- *Ai-Te Kuo, Wei-Shinn Ku, and Xiao Qin are with the Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849 USA. E-mail: {aite, weishinn, xqin}@auburn.edu.*
- *Haiquan Chen is with the Department of Computer Science, California State University, Sacramento, CA 90032 USA. E-mail: haiquan.chen@csus.edu.*
- *Liang Tang is with the Facebook, Menlo Park, CA 94025 USA. E-mail: liang.xjtu@gmail.com.*

Manuscript received 25 April 2021; revised 12 November 2021; accepted 29 January 2022. Date of publication 16 February 2022; date of current version 3 April 2023.
This work was supported in part by the U.S. National Science Foundation under Grants IIS-1618669 (III) and ACI-1642133 (CICI).
(Corresponding author: Ai-Te Kuo.)
Recommended for acceptance by B. Glavic.
Digital Object Identifier no. 10.1109/TKDE.2022.3151740

1041-4347 © 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:38:27 UTC from IEEE Xplore. Restrictions apply.

TABLE 1
Example of Hotel Ratings

Hotel	Inst.	Prob.	Amenities	Location	Price	Service
A	a_1	0.6	5	3	5	4
	a_2	0.2	3	2	4	5
	a_3	0.2	4	3	4	4
B	b_1	0.3	3	3	3	4
	b_2	0.4	2	3	4	3
	b_3	0.3	3	4	3	4
C	c_1	0.5	3	2	3	3
	c_2	0.2	2	4	4	4
	c_3	0.3	4	3	4	3

scenarios in the real world. For example, Alice is looking for a hotel, and each hotel has multiple reviews with multi-criteria ratings. In the search for a hotel, amenities, location, room price, room service, and room cleanliness are some of the criteria to consider (See Table 1). In this multi-criteria decision-making problem, the probabilistic skyline queries can present all the hotels of interest, i.e., hotels that are not worse than any other hotel in all the criteria. An exact p-skyline query returns (1) all the hotels taking a probability of at least p to be in the skyline and (2) the exact skyline probabilities of the skyline hotels. Here each hotel can be treated as an object, and each instance of the objects corresponds to a review associated with a probability (confidence value). A hotel is in the skyline if it is not dominated by any other hotel, i.e., there does not exist a hotel that has a better rating in every criterion.

Our Contributions. In this paper, we study distributed evaluation of the exact p-skyline queries on uncertain data using MapReduce for high efficiency and scalability. Our contributions can be summarized as follows:

- We devise a novel MapReduce-based framework, *ProbSky*, in support of distributed evaluation of the probabilistic skyline queries. ProbSky is adept at efficiently evaluating the exact p-skyline queries on large uncertain data without compromising the quality of query results.
- We formally prove two pruning lemmas and propose dominant instance pruning (DIP) by leveraging the most dominant instances to rule out unqualified objects in the early stage.
- We develop slab-based partitioning (SP) to minimize communication costs and evenly distribute workload across compute nodes.
- We employ reference point-based acceleration (RPA) to evade avoidable dominance tests, accelerating the computation of the skyline probabilities.
- Extensive experiments using both real and synthetic datasets verify that compared to the state-of-the-art methods, ProbSky was able to speed up the evaluation of the exact p-skyline queries by at least one order of magnitude in most cases. Our experiments also validate that ProbSky managed to alleviate the system bottlenecks that may cause severe performance deterioration by balancing memory consumption and workload among different compute nodes in distributed environments.

The rest of this paper is organized as follows. Section 2 surveys the related work. Section 3 provides the problem formulation. Section 4 elaborates on ProbSky while Section 5 details the three optimization techniques for performance improvement. Section 6 compares the performance of ProbSky with the state-of-the-art methodologies. Section 7 concludes the paper.

2 Related Work

Skyline Queries. During the past decade, skyline queries [2] have been extensively studied and applied in various decision-making applications. Methods proposed to address skyline query processing include Bitmap [21], Sort Filter Skyline [1], Lattice Skyline Algorithm [22], Branch-and-Bound Skyline [23], Object-Based Space Partitioning Scheme [24], Z-Sky [25], and MBR-Oriented Approach [26].

Skyline Queries using MapReduce. MapReduce [27] was first introduced by Google in 2004 for large-scale data-intensive processing on distributed platforms. Zhang et al. [28] piloted MapReduce for parallel skyline queries. In an angle-based space partitioning scheme over multi-dimensional space [29], data points were transformed from Cartesian coordinates to hyperspherical coordinates, and then the space was partitioned by using hyperspherical coordinates. Chen et al. [30] introduced an angular partitioning approach to MapReduce-based skyline queries. Zhang et al. [31] devised a two-phase MapReduce framework for parallel skyline evaluations. The framework utilizes angle-based partitioning for pruning in the first phase and grid-based partitioning for computing global skyline points in the second phase. A similar partitioning scheme was proposed by Köhler et al. [32] that partitions the space by hyperplane projections. Recently, Wijayanto et al. [33] designed MapReduce-based parallel skyline query processing using L-shaped partitioning for high-dimensional large-scale datasets.

*Probabilistic Skyline Queries.*Compared to traditional skyline queries, the probabilistic skyline query is a popular and powerful paradigm for extracting decisive information from uncertain databases. The concept of evaluating the probabilistic skyline queries was first introduced by Pei et al. [8], followed by the development of two algorithms, bottom-up and top-down, using R-tree to prune the search space based on the assumption that each instance carries the same probability to happen. Atallah et al. [9] investigated a sub-quadratic complexity algorithm, relying on space partitioning approaches and the dominance counting algorithm [34]. Kim et al. [11] implemented a novel way of applying Z-tree to find as many incomparable groups of instances as possible to avoid unnecessary dominance tests. Lian and Chen [35] studied monochromatic and bichromatic probabilistic reverse skyline queries on uncertain data. Nevertheless, all the above-mentioned belong to the category of serial algorithms failing to handle demanding scenarios where data volume is persistently growing.

Probabilistic Skyline Queries using MapReduce. The MapReduce framework [16] has been extensively incorporated into probabilistic skyline computation [31], [36], [37], [38]. For example, Ding et al. [36] proposed a two-phase filter-refine approach under the MapReduce framework (PSMR). Recently,

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:38:27 UTC from IEEE Xplore. Restrictions apply.

Gavagsaz et al. [39] proposed one-pass MapReduce-based probabilistic skyline queries (PCPS) by applying smart sampling of points from the critical sections and grid partitioning for workload balancing. However, they assume that every tuple in the uncertain database is attached with an occurrence probability, which can be regarded as a bivariate distribution rather than a multivariate distribution in state-of-the-art studies [8], [9], [10], [11], [37]. That is, PSMR and PCPS are a special case of the probabilistic skyline queries where each uncertain object is associated with exactly one instance and an occurrence probability, and thus cannot process the probabilistic skyline queries with uncertain objects described by joint probability mass and density functions.

Park et al. [37] presented two approaches to the probabilistic skyline queries under the MapReduce framework, PS-QPF-MR and PS-BRF-MR. The algorithms initially construct an in-memory quadtree [40] by random sampling and broadcast the quadtree to the worker machines, each of which filters out unqualified objects based on the characteristics of leaf nodes. The sampling and the choice of the split threshold determine the formation of the quadtree, which indirectly affects the performance. An error resulting from an incorrect choice of the split threshold or a skewed sampling distribution may cause a single leaf node associated with bulky data hardly fitting into main memory when coping with high-volume uncertain data. Moreover, the adopted in-memory quadtree structure tends to create skewed leaf nodes resulting in a skewed imbalance in workload, thereby leading to poor scalability. Nevertheless, both PS-BRF-MR and PS-QPF-MR assume all the instances of an object must be on the same machine, making it unrealistic to conduct redistribution of the entire dataset across machines for each query evaluation in a dynamically changing database environment. In this paper, we delve into the development of a novel MapReduce-based framework, ProbSky, for scalable probabilistic skyline evaluation on large uncertain data. ProbSky is adroit at efficiently evaluating the exact p-skyline queries on large uncertain data without compromising the quality of query results. A salient feature of ProbSky is that the instances of the same object are spread across multiple machines in a cluster computing environment.

3 Problem Formulation

In what follows, we elaborate on the problem statements for the skyline queries and the probabilistic skyline queries. A summary of the symbols is presented in Table 2.

3.1 Skyline Queries

Given a d-dimensional dataset D, a data point $p = (p_1, p_2, \ldots, p_d)$ is said to dominate, denoted by $\prec$, another point q if and only if $\forall k \in K = \{(\forall x \in N, p_x \leq q_x)$ and $(\exists x \in N, p_k < q_k) \,|\, 1 \leq x \leq d\}$; conversely, $p \not\prec q$ denotes $(\exists k \in K, p_k > q_k) \vee (\forall k \in K, p_k = q_k)$. The elements of any dominance relationship can be extended to a group. For example, $p \prec G$ denotes p dominates every point in G, $G \prec p$ denotes every point in G dominates p, and $G_1 \prec G_2$ signifies that each point in G_1 dominates all points in G_2. A point p in D is a skyline point if it satisfies $\neg\exists p \in D, p \prec q \in D$. A skyline query returns a set of all skyline points.

TABLE 2
Summary of the Symbols Used

Symbol	Signification
OS	Data space, $OS = \{OS_1, \ldots, OS_n\}$
U, V	Arbitrary uncertain object
u, v	Arbitrary instance of uncertain object U, V
$u \prec v$	Instance u dominates instance v
I_c	The candidate instances
I_f	The influential instances
I_d	The most dominant instances
R	Arbitrary reference point
$H(R)$	The hyper-rectangle of reference point R
$P(u)$	The occurrence probability of instance u
p	The probability threshold
$S^*(U), S^*(u)$	The approximate skyline probability of U, u
$S(U), S(u)$	The exact skyline probability of U, u
$PS(OS, p)$	The skyline objects with respect to OS and p

3.2 Probabilistic Skyline Queries

Given a set of uncertain objects OS and a probability threshold p $(0 \leq p \leq 1)$, the overarching goal of the probabilistic skyline queries is to compute the skyline probability $S(U)$ of every uncertain object $U \in OS$ (i.e., the likelihood uncertain object U becomes a skyline object).

An uncertain object $U \in OS$ can be described by a continuous or a discrete model. In a continuous model, U is described by a continuous PDF. In a discrete model, U is represented by a set of instances $U = \{u_1, \ldots, u_n\}$, each of which is a d-dimensional vector following the discrete PDF of U. In this paper, we mainly focus on the discrete model of the probabilistic skylines. Following [8] [37], we assume that the occurrence probability $P(u)$ of an instance u only depends on the object's PDF (i.e., the objects are independent of each other).

Given two uncertain objects U and V, the probability of an instance $u \in U$ being dominated by V is formalized as follows.

$$P(V \prec u) = \sum_{v \in V,\, v \prec u} P(v) \tag{1}$$

Based on the law of total probability, the probability that i is a skyline instance with respect to V (i.e., instance i is not dominated by V) can be rewritten as follows.

$$P(V \not\prec u) = 1 - \sum_{v \in V,\, v \prec u} P(v) \tag{2}$$

The probability of instance i not dominated by any other objects can be derived as follows.

$$S(u) = \prod_{V \in OS - U} P(V \not\prec u) = \prod_{V \in OS - U} \left(1 - \sum_{v \in V,\, v \prec u} P(v)\right) \tag{3}$$

Thus, object U not dominated by any other objects can be derived below.

$$S(U) = \sum_{u \in U} P(u) \prod_{V \in OS - U} \left(1 - \sum_{v \in V,\, v \prec u} P(v)\right) \tag{4}$$

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:38:27 UTC from IEEE Xplore. Restrictions apply.

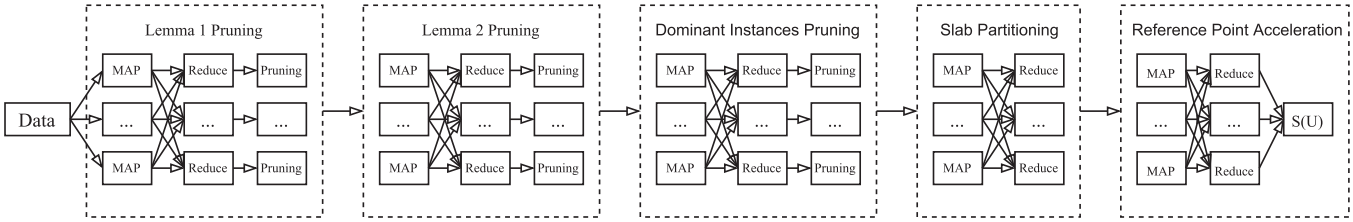


Fig. 1. Illustration of ProbSky architecture.

Finally, the result of a probabilistic p-skyline query with respect to OS is formalized as follows.

$$PS(OS, p) = \{U \in OS \mid S(U) \geq p\} \tag{5}$$

Take hotel A in Table 1 as an example where a lower value is considered better. As the skyline probability of an uncertain object is the sum of each of the instance skyline probabilities multiplied by the likelihood each instance will occur, the instance skyline probabilities shall be computed first based on Eq. (3). Point a_1 is dominated by points b_1, b_2, c_1, c_3, thereby $S(a_1) = (1 - 0.3 - 0.4) \times (1 - 0.5 - 0.3) = 0.06$. Point a_2 is dominated by points c_1, thereby $S(a_2) = (1 - 0) \times (1 - 0.5)) = 0.5$. Point a_3 is dominated by points b_1, b_2, c_1, c_3, thereby $S(a_3) = (1 - 0.3 - 0.4) \times (1 - 0.5 - 0.3) = 0.06$. According to Eq. (4), hotel A's object skyline probability is derived $S(A) = 0.6 * 0.06 + 0.2 * 0.5 + 0.2 * 0.06 = 0.148$. Similarly, $S(B) = 0.7$ and $S(C) = 0.8$. Given a probability threshold p of 0.6 and an uncertain object set OS of hotels A, B and C, the probabilistic query $PS(OS, p)$ returns B and C.

4 *ProbSky*

In this section, we first paint an overview of ProbSky followed by elaborating on the two stages - pruning and computing - in ProbSky.

4.1 ProbSky Overview

The architecture of *ProbSky* is illustrated in Fig. 1. ProbSky consists of two stages, namely, pruning and computing.

- *Pruning Stage* We first filter out unqualified skyline instances using Lemma 1. Next, Lemma 2 is utilized to prune the uncertain objects of the remaining instances, and the most dominant instances are collected. After that, the uncertain objects with the skyline probabilities lower than the threshold are pruned using the dominant instance pruning (DIP).
- *Computing Stage* We employ the slab-based partitioning (SP) to evenly distribute the candidate instances and the respective influential instance set across worker machines to maximize parallelism and achieve workload balancing. Subsequently, the partial object skyline probabilities are computed using reference point-based acceleration (RPA). Finally, the object skyline probabilities are obtained by the aggregation of the partial object skyline probabilities from all the partitions, and the objects with the skyline probabilities lower than the threshold are eliminated.

4.2 Pruning Stage

ProbSky builds on two pruning lemmas to effectively and efficiently determine the candidate instance set and the influential instance set for calculating the skyline probabilities of the candidate objects at the computing stage.

4.2.1 Collecting Influential Instances

Lemma 1. *Given an instance i, if any of the object's maximum corners $U_{max} \prec i, U \in OS, i \notin U$, then $S(i) = 0$.*

Proof. Assume every dimension value of every instance contains non-negative numerical values. Given an uncertain object U, a d-dimensional hyper-rectangle $R(U)$ can be created to contain all the instances of U. The lower-left corner of $R(U)$ is at the origin, and the upper-right corner of $R(U)$ is at a d-dimensional point $(\max_{i \in U} i.p_1, \max_{i \in U} i.p_2, \ldots, \max_{i \in U} i.p_d)$, where $i.p_n$ is the nth dimension value of an instance i. In the rest of the paper, we refer to the upper-right corner of $R(U)$ as object U's maximum corner, denoted by U_{max}.

As U_{max} is a point at the maximum values of each dimension of U, the geometric properties of U_{max} show that if $U_{max} \prec i$, then every instance of U dominates i, denoted by $U \prec i$.

From Eq. (3), it is clearly seen that if $\forall v \in V, v \prec u$, one of the series of factors will be zero because $\sum_{v \in V,\, v \prec u} P(v) = 1$, hence $S(i) = 0$.

The influential instances are instances not dominated by any of the object maximum corners; thus, the preceding job is to obtain every object's maximum corner U_{max}. As the instances are distributed across the worker nodes by the cluster manager, the instances of an object may reside across worker nodes. Thus, to obtain the maximum corners of the uncertain objects, we apply a set of key-value pairs for parallel reductions where *key* is an object identifier, and *value* is the maximum corner of the object. After all the maximum corners of the uncertain objects are obtained, they are broadcast to each worker node for Lemma 1 pruning. However, most of the objects' maximum corners barely contribute to Lemma 1 pruning as the maximum corner of an uncertain object is a set of the maximum values of each dimension among its instances, hence typically not dominant. Thus, dominance testing between every object's maximum corner and every instance is inefficient.

To address this issue, we employ a random sample to effectively and efficiently find the maximum corners of the uncertain objects with the highest pruning power. First, a random sample of size c is drawn from the population of the instances to be an unbiased representation of data. Subsequently, each instance in the sample performs dominance tests with every object's maximum corner. Only the most k

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:38:27 UTC from IEEE Xplore. Restrictions apply.

dominant maximum corners OC are retained for pruning non-contributory instances, and thus the time complexity narrows down from $\mathcal{O}(XY)$ to $\mathcal{O}(cX + X\log X + kY)$ where X is the total number of uncertain objects, Y is the total number of instances. After being collected, OC is broadcast to every worker node, each of which starts pruning by Lemma 1. The instances being ruled out by Lemma 1 are non-contributory instances, hence disregarded, and the rest of the instances comprise influential instances I_f, which can also be formalized as $I_f = \{p \in I \mid \nexists U \in OS, U_{max} \prec p\}$.

Here we present the pseudo-code of the Lemma 1 pruning in Algorithm 1, corresponding to the leftmost dotted box in Fig. 1. The *RUN* function is executed on the master node, and the *Pruning* function is broadcast to each worker node, which is requested to rule out unqualified probabilistic skyline instances. Lines 1-9 show the process of Lemma 1 pruning. Once an instance is dominated by one of the objects' maximum corners, it has no effect on the result; thus, the instance can be disregarded without examining the following corners. Line 11 takes a random sample of instances from the global object set and then broadcasts the sample to all the worker nodes. Line 12 first collects all the objects' maximum corners across the worker nodes, computes each corner's dominant power in S, and retains the top k object corners with the highest pruning power in R. Line 13 broadcasts the chosen object corners and the pruning function to the worker nodes and requests them to prune non-contributory instances.

Algorithm 1. Lemma 1 Pruning

```
1:  function PRUNING(OS, OC)
2:      for all instance ∈ OS as i do
3:          flag ← True
4:          for all object corner c ∈ OC do
5:              if c ≺ i then
6:                  flag ← False
7:                  break
8:          if flag then
9:              emit i
10: function RUN(OS, k)
11:     S ← Random Sample of Instances from OS
12:     OC ← Top k Dominant Object Corners in S
13:     I_f ← PRUNING(OS, OC)
14:     return I_f                          ▷Influential Instance Set
```

4.2.2 Collecting Candidate Instances

Lemma 2. *Given an uncertain object $U \in OS$ and a probability threshold p, if $S^*(U) < p$, then U is an unqualified object, and all the instances of U can be safely eliminated from the candidate instance set I_c.*

Proof: Let OS_i be the uncertain object set with respect to partition i, $S_i(U)$ be the partial skyline probability of object U with respect to partition i, and n be the number of partitions. The partial skyline probability of object U in partition i can be formalized as follows.

$$S_i(U) = \sum_{U \in OS_i} \sum_{u \in U} P(u) \prod_{V \in OS - U} \left(1 - \sum_{v \in V, v \prec u} P(v)\right) \tag{6}$$

Then, each partition randomly divides the instances in the partition into equal-sized bins B; then, each instance u in a bin $B(u)$ only compares to the instances in the same bin and omits dominance tests from other bins. Let $S_i^*(U)$ be the result from the aggregation of all the bins in partition i with respect to object U. Since the inequality $0 \le \sum_{v \in V, v \prec u} P(v) \le 1$ holds and $OS - U$ is a super set of all the bins, the inequality is obtained as follows.

$$\begin{aligned} S_i^*(U) &= \sum_{U \in OS_i} \sum_{u \in U} P(u) \prod_{V \in B(u) - U} \left(1 - \sum_{v \in V, v \prec u} P(v)\right) \\ &\ge \sum_{U \in OS_i} \sum_{u \in U} P(u) \prod_{V \in OS - U} \left(1 - \sum_{v \in V, v \prec u} P(v)\right) \\ &= S_i(U) \end{aligned} \tag{7}$$

$$S^*(U) = \sum_{i=1}^{n} S_i^*(U) \ge \sum_{i=1}^{n} S_i(U) = S(U) \tag{8}$$

Finally, $S^*(U)$ is obtained by the aggregation of all the partial skyline probabilities from every bin. From Eq. (8), we show that $S^*(U)$ is higher than or equal to $S(U)$. Therefore, object U with $S^*(U) < p$ can be removed from the candidate object set, the instances of which are called candidate instances I_c.

Algorithm 2. Lemma 2 Pruning

```
1:  function L2_MAP(I_f)
2:      S ← Equal-sized bins of OS
3:      P ← new HashMap                          ▷S*
4:      for all bin B ∈ S do
5:          for all instance i ∈ B do
6:              T = new HashMap
7:              for all instance j ∈ B do
8:                  if j ≺ i and i.obj_id ≠ j.obj_id then
9:                      T[j.obj_id] ← T[j.obj_id] + j.prob
10:             P[i.obj_id] ← P[i.obj_id] + i.prob × ∏_{p∈T}(1 − p)
11:     emit P
12:   function L2_REDUCE(P_1, P_2)
13:     emit Merge P_1 and P_2 by addition
14:   function RUN(I_f, t)
15:     A ← L2_MAP(I_f).REDUCE(L2_REDUCE)
16:     A ← {A.key | A.value ≥ t}
17:     I_c ← {i | i ∈ I_f, i.obj_id ∈ A}
18:     return I_c
```

Lemma 2 pruning is executed after I_f is collected. Each worker node divides the influential instances in the partition into equal-sized bins, followed by calculating each instance skyline probability with respect to the locating bin $S^*(u)$, and finally aggregates all of them by object to approximate the object skyline probabilities $S^*(U)$. By Lemma 2, all the instances of an uncertain object U with $S^*(U) < p$ can be eliminated, and then the rest of the instances become candidate instances.

Algorithm 2 presents the pseudo-code of Lemma 2 pruning, as the second dotted box in Fig. 1. Lines 1-11 describe the process of collecting the approximations of the object skyline probabilities. Lines 12-13 display how the approximations are reduced across partitions. Entries with

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:38:27 UTC from IEEE Xplore. Restrictions apply.

duplicate keys will be merged into one entry with the sum of values. Lines 14-18 present the execution code run on the master node. It first computes the partial approximations of the object skyline probabilities, aggregates all the approximations by object, excludes unqualified objects by the approximations, and finally emits the instances of the candidate uncertain objects.

After the pruning stage is accomplished, the candidate instance set I_c and the influential instance set I_f are originated. Next, we compute the exact skyline probabilities of the uncertain objects. The naive idea here is to evenly distribute I_c, broadcast I_f to all the worker nodes, and eventually compute the object skyline probabilities. Unfortunately, dominance tests between I_c and every instance in I_f are inefficient. To address this issue, we propose a dynamic mapping strategy to minimize inter-worker communication while balancing workload governed by slab-based partitioning (see Section 5.2).

5 Optimization Techniques

In this section, we shed a bright light on the three optimization techniques: dominant instance pruning, slab-based partitioning, and reference point-based acceleration. All three techniques are seamlessly integrated into ProbSky to enhance efficiency and scalability.

5.1 Dominant Instance Pruning

According to Eq. (7), the set of the most dominant instances I_d can be regarded as a bin and be broadcast to all the worker nodes, each of which applies Lemma 2 to rule out unqualified objects. However, a key challenge here is how to effectively select the most dominant instances I_d. A naive idea is to utilize a min-heap to maintain I_d by the sum of squares of dimension values. However, this approach works poorly when the dimension boundaries vary greatly. We propose to use the occurrences of dominance, domination power, as the criterion to select the most dominant instances. When any dominance occurs during the computation of the approximations, the domination power of the dominating instance increments. After the completion of the sampling-based approximating, only the top instances with the highest domination power are retained. Next, we pick the top-k most dominant instances. Subsequently, the top-k most dominant instances are broadcast to all the worker nodes. Each worker node examines each instance with the most dominant instances and estimates the object skyline probabilities. Finally, according to Lemma 2, the instances of the uncertain objects with $S^*(U) < p$ are excluded from I_c. This technique is applied in the third dotted box shown in Fig. 1, and the algorithm of this technique is similar to Algorithm 2 but with S being a single bin of I_d and B in the outer-loop being replaced by the set of the instances emitted by Lemma 2 pruning.

5.2 Slab-Based Partitioning for Workload Balancing

A naive idea to compute the skyline probabilities of the uncertain objects is to evenly distribute I_c along with I_f. Dominance tests thus need to be checked between every instance in I_c and every instance in I_f. According to Eq. (4), the skyline probability of an uncertain object U depends only on the influential instances that dominate the instances of U. Thus, for a given subset of the candidate instances, the objective is to minimize the cardinality of the set of instances needed to be checked without compromising the quality of query results.

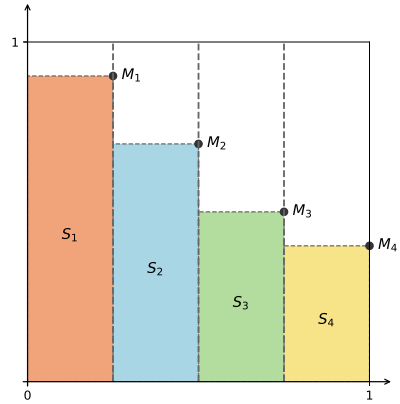


Fig. 2. Illustration of the slab-based partitioning for workload balancing.

Here we present a dynamic mapping strategy to allocate data to the target machines using slab-based partitioning for achieving workload balancing. Recall that we collect two sets of data during the pruning stage, the candidate instance set I_c (the instances of the candidate uncertain objects) and the influential instance set I_f. First, I_c is sorted by ascending order of a dimension, x axis in default, and then distributed to n equal-sized bins along the x axis where n is the number of reducers. Consequently, each bin of the instances composes a slab. Take Fig. 2 as an example. S_3 is a slab denoted by green color and contains a quarter of all the instances.

Aside from O_C, partial influential instances from I_f also need to be distributed to the target reducer. Given one slab collecting P_c, instances that have a possibility of dominating at least one instance in P_c lie in the region from the origin to the upper-right corner of the slab. As depicted in Fig. 2, M_3 is the upper-right corner of slab S_3, and thus the instances in the area from the origin to M_3 are shipped to the target reducer as the respective influential instances. The process is repeated until every reducer obtains the respective candidate instance set and influential instance set. After data are mapped to the target reducers, the reduce function starts with the iteration of every candidate instance in I_c. Each candidate instance computes its skyline probability with respect to the respective influential instances, and then each instance skyline probability times its probability is outputted to a set of key-value patterns for further aggregation where *key* is an object identifier, and *value* is the object skyline probability. After the set of key-value patterns is harvested, all the object skyline probabilities $S(U)$ are obtained by an addition-aggregation by object. Finally, p-skyline object set is derived by retrieving objects U whose skyline probability $S(U)$ is larger than or equal to p.

Let $M = \{M_1, \ldots, M_n\}$ be the total number of candidate instances, where M_i is the number of candidate instances in reducer i, and N_i be the number of influential instances in reducer i. The computing stage computes all the object skyline probabilities with a two-level nested loop. Thus, the time complexity of the computing stage with respect to

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:38:27 UTC from IEEE Xplore. Restrictions apply.

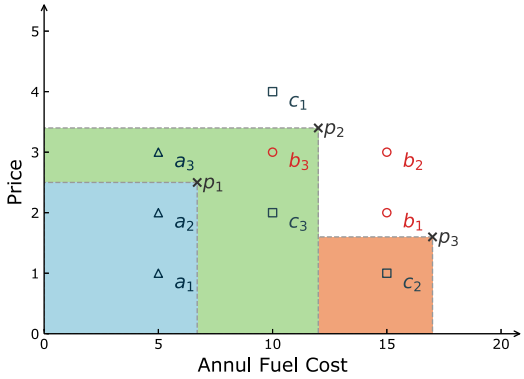


Fig. 3. Illustration of the reference point-based acceleration to reduce the number of dominance tests in a 2-dimensional space.

reducer i is $\mathcal{O}(M_i N_i)$. As the final result relies on all the reduce tasks, the overall performance is determined by the last completed task. Thus, the time complexity is $\max_{i=1}^{n} M_i N_i$. Here we evenly distribute the sorted candidate instances to each reducer, and thus each of them has the same number of candidate instances. It is worth mentioning that the number of influential instances in the reducer determines its workload and the one with the highest number of influential instances becomes a performance bottleneck. Our goal here is to have each reducer carry the same amount of workload (i.e., $M_1 N_1 = M_2 N_2 = \ldots = M_n N_n$). Next, we obtain $M_1 K(M_1) = M_2 K(M_1 + M_2) = \ldots = M_{n-1} K(M_1 + \ldots + M_{n-1}) = M_n K(M_1 + \ldots + M_n)$. Then, the ratio of $M_1, M_2, \ldots, M_n$ is computed. Finally, given $M_1 + M_2 + \ldots + M_n = M$, the number of candidate instances with respect to reducer i can be derived to strike a balanced workload by evenly distributing candidate objects on multiple machines.

5.3 Reference Point-Based Acceleration

To further reduce the unnecessary dominance tests, we employ reference points to accelerate the computation of object skyline probabilities. Take Fig. 3 as an example. Let us consider three reference points p_1, p_2, and p_3, corresponding to three regions, blue, green, and tangelo, respectively. One interesting observation is that each point in a region is not dominated by any point outside of its region (i.e., the points outside of the region are all non-contributory to the points inside the region). For example, a_1 and a_2 are in the blue rectangle, and the points possible to dominate a_1 and a_2 can only appear in the same rectangle. In this case, a_1 needs no examination because there is no point from any other objects in the blue rectangle. For c_2, we only need to check one point a_1. For b_3 we only need to check a_1, a_2, a_3 and c_3 while the other two points outside of the rectangle, c_1 and c_2, can be disregarded.

A reference point R is a d-dimensional data point and forms a d-dimensional hyper-rectangle $H(R)$ along with the origin. If a point is dominated by R, it is dominated by all the instances in $H(R)$. Conversely, if any point dominates R, the possible points that dominate the point fall in the hyper-rectangle $H(R)$. Thus, given a data point p, the task is to spot any of the hyper-rectangles $H(R)$ that contains p and examine only the data points inside $H(R)$, leading to a reduced number of dominance tests. Notwithstanding the concept, inappropriate selection of reference points might provide little improvement or even performance degradation. To refine the selection of reference points, a variant of hyperoctrees is proposed to find a set of promising reference points that efficiently and effectively help to cut off the number of instances to examine by leveraging the geometric properties.

The selection of promising reference points is based on a variant of d-dimensional region-based hyperoctrees [40]. We develop a new variant of hyperoctrees, lazy hyperoctrees, to help decide a set of promising reference points without generating a plethora of empty leaf nodes. Inspired by lazy evaluation, the idea is that a child node is added to the children of the parent node only when it is needed. In the implementation of a lazy hyperoctree, each node is associated with an orthant, specified by the bounding box of the points inside. A child node is only added to the parent node when the respective orthant contains more than the threshold number of points m. Unlike any other hyperoctrees, information associated with the leaf nodes is of no interest in the lazy hyperoctree, and thus the leaf nodes can be seen virtual, never created. The creation of a lazy hyperoctree follows the principle of the recursive decomposition where the data space is recursively divided by d mutually perpendicular coordinate hyperplanes into 2^d equal parts until the number of points inside the orthant does not exceed m. We refer to each of the 2^d orthants as a child orthant to avoid confusion. As each child orthant can be described by its signs of the coordinates, a d-bit string is used to represent the index of a child orthant where each bit corresponds to the sign of a coordinate. The bit can take two values: 0 indicates that the sign of the coordinate of the orthant is negative (-) while 1 indicates positive (+).

Initially, the parent node evaluates the total number of points in each child orthant that contains points. As each point only corresponds to one child orthant, the number of child orthants is at most n rather than 2^d where n is the number of points in the orthant. Then, the algorithm iterates over each of the child orthants to check whether a node associated with the visiting child orthant should be added to the children of the parent node. If the addition of the node is needed, then the algorithm calculates the bounding box of the points inside the child orthant and adds the node with the bounding box as its orthant.

We create a lazy hyperoctree for each partition. Given a partition with a subset of the candidate instances $P_c \subseteq I_c$ and a set of the respective influential instances $P_f \subseteq I_f$, a lazy hyperoctree is created from the points of P_c, after which the upper-right corners of the orthant of every internal node form a set of reference points. Then, each reference point R in the set iterates over P_f to hold a set of instances whose points lie inside the hyper-rectangle $H(R)$. As every candidate instance $u \in P_c$ is assigned to a node upon the creation of the hyperoctree, one of the containing hyper-rectangles is known, and so is the set of instances for u to test $I_t \subseteq P_f$. Subsequently, dominance relationships between u and I_t are examined to compute the instance skyline probabilities $S(u)$, after which all of them are aggregated to derive the object skyline probabilities $S(U)$ by object. According to Eq. (5), the candidate uncertain objects with $S(U) < p$ are eliminated, and the remaining uncertain objects form the skyline object set with respect to OS and p. Note that this

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:38:27 UTC from IEEE Xplore. Restrictions apply.

technique can also be applied to expedite the computation of $S^*(U)$ in Lemma 2 pruning and the dominant instance pruning.

Algorithm 3 presents the pseudo-code of the reference point-based acceleration and is applied in the rightmost dotted box in Fig. 1. The threshold number of points for a child node to be added is m. Lines 2-3 create a lazy hyperoctree and hyper-rectangles using the internal nodes of the hyperoctree. Lines 4-12 describe how to compute the object skyline probabilities with the lazy hyperoctree and the hyper-rectangles. Function *BuildLazyTree* creates a lazy hyperoctree following the above-mentioned procedure. Function *BuildHyperRect* constructs the hyper-rectangles using the reference points and, for each hyper-rectangle, holds a subset of P_f that lie inside the hyper-rectangle. Given a hyper-rectangle, function *getInstancesInHyperRectangles* returns all the instances inside the hyper-rectangle. Function *RPA_Reduce* is exactly the same as *L2_Reduce* that merges the emitted object-probability tables.

Algorithm 3. Reference Point Acceleration

1: **function** RPA_MAP(P_c, P_f)
2: $T \leftarrow$ *BuildLazyTree*(P_c, m)
3: $H \leftarrow$ *BuildHyperRect*(T.*internalNodes*, P_f)
4: $P \leftarrow$ *new HashMap*
5: **for all** *candidate instance* $u \in P_c$ **do**
6: $T \leftarrow$ *new HashMap*
7: $R \leftarrow u$.*node.upperRightCorner*
8: $I_t \leftarrow$ *getInstancesInHyperRectangles*($H(R)$)
9: **for all** *influential instance* $j \in I_t$ **do**
10: **if** $v \prec u$ **and** $u.obj_id \neq v.obj_id$ **then**
11: $T[v.obj_id] \leftarrow T[v.obj_id] + v.prob$
12: $P[u.obj_id] \leftarrow P[u.obj_id] + u.prob \times \prod_{p \in T}(1-p)$
13: **emit** P
14: **function** RUN(I_c, I_f)
15: $A \leftarrow$ RPA_MAP(I_c, I_f).REDUCE(RPA_REDUCE)
16: $A \leftarrow \{A.key \mid A.value \geq \mathrm{t}\}$
17: **return** A

6 Experimental Validation

In this section, we empirically demonstrate the superiority of ProbSky over state-of-the-art methods. The research began by comparing methods, followed by performance comparisons, driven by the real-world and synthetic datasets, respectively.

6.1 Comparison Methods

The following methods were implemented using Spark 3.0.1 in Apache Hadoop 2.7.6. The experiments were conducted on an AWS cluster with one master node and eight shared-nothing workers. The master acted as a cluster resource manager. Each node was an Amazon EC2 R6g instance with 8 GB of RAM running on 64bit Amazon Linux 2 AMI. All the nodes were connected through the 10 Gigabit Ethernet network. Each set of experiments in this section was repeatedly carried out twenty times and the averaged results were reported. Note that PS-QPF-MR and PS-BRF-MR are referred to as QPFMR and BRFMR in the experiment tables, respectively.

- *PS-BRF-MR* [37]: The parallel probabilistic skyline query algorithm utilizes a quadtree for pruning and computes object skyline probabilities based on random partitioning.
- *PS-QPF-MR* [37]: The parallel probabilistic skyline query algorithm applies a quadtree for pruning and computes object skyline probabilities based on quadtree partitioning.
- *ProbSky*: The complete version of the proposed ProbSky embraces all three optimization techniques.
- *PSky-D*: This is a degraded version of ProbSky with the dominant instance pruning removed.
- *PSky-S*: In this degraded version of ProbSky, the slab-based partitioning module is removed.
- *PSky-R* This degraded version of ProbSky lacks the reference point-based acceleration module.
- *PSky-V*: The vanilla version of ProbSky with only Lemma 1 and Lemma 2 pruning techniques.

6.2 Performance Comparison on Real Data

The performance edge of *ProbSky* was quantitatively evaluated over the existing solutions (PS-BRF-MR and PS-QPF-MR) from three perspectives, namely, overall efficiency (see Section 6.2.1), memory consumption (see Section 6.2.2), and execution time per machine (see Section 6.2.3).

We carried out extensive experiments driven by the TripAdvisor dataset [41], which comprised a total of 50,264,531 reviews for 365,056 hotels. Each review consisted of one mandatory overall rating and eight optional sub-ratings (e.g., room rating, location rating, cleanness, etc). In our experiments, each hotel was regarded as an object while the reviews of the hotel were treated as its instances. We created two datasets (TripAdvisor-4d and TripAdvisor-7d) derived from the reviews with four and seven ratings, respectively. Each value of a review rating was injected with a noise by using a Gaussian distribution with zero mean and a standard deviation of 0.05. Unless otherwise specified, the cardinality was 100K; the number of instances per object was 7, the number of machines was 8, the probability threshold was 0.6, the number of dominant instances was 2,000, and the threshold value of the lazy hyperoctree was 5.

6.2.1 Comparison for Efficiency

To demonstrate the overall performance strengths, we compared ProbSky against PS-BRF-MR and PS-QPF-MR in four groups of experiments. These groups of experiments made use of real-world data to gauge the impacts of the probability threshold, the number of machines, the number of objects, and the number of instances on the efficiency.

Impact of Probability Threshold. Recall Algorithms 2 and 3 (see Section 5) are reliant on a critical system parameter - probability threshold. This group of experiments was dedicated to quantifying the effect of the probability threshold on the overall performance of ProbSky and the competitors. Table 3 shows the impact of the probability threshold on runtimes for all the tested methods. As depicted in Fig. 3, ProbSky consistently outperformed all the competitors on both TripAdvisor-4d and TripAdvisor-7d datasets in terms of query efficiency in most cases except when the probability threshold was 0.8 on the TripAdvisor-4d dataset. In that

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:38:27 UTC from IEEE Xplore. Restrictions apply.

TABLE 3
Varying Probability Threshold

Dataset	TripAdvisor-4d				TripAdvisor-7d			
	0.2	0.4	0.6	0.8	0.2	0.4	0.6	0.8
ProbSky	**35.4**	**31.6**	**27.2**	26.0	**61.4**	**45.8**	**40.1**	**35.2**
PSky-S	37.3	31.8	28.5	27.1	64.7	48.3	42.4	36.5
PSky-D	51.1	34.2	29.1	26.8	140.3	98.1	68.3	48.3
PSky-R	37.2	31.6	29.5	25.9	94.8	61.1	45.9	38.3
PSky-V	71.6	43.2	28.7	**23.9**	312.0	254.2	206.6	127.9
QPFMR	74.0	46.8	39.7	38.2	432.4	308.6	210.9	152.5
BRFMR	2268.7	2201.0	2083.6	1781.0	2582.1	2337.5	2224.2	2100.0

TABLE 4
Varying Number of Machines

Dataset	TripAdvisor-4d				TripAdvisor-7d			
	4	8	16	32	4	8	16	32
ProbSky	**28.0**	**27.1**	22.3	22.6	**38.4**	**40.0**	**29.7**	**29.6**
PSky-S	28.5	28.5	23.1	22.0	39.2	42.4	31.9	30.0
PSky-D	29.2	29.1	23.7	23.2	97.2	68.3	46.2	41.7
PSky-R	29.4	29.5	23.2	22.9	50.7	45.9	32.1	29.9
PSky-V	28.5	28.7	**22.0**	**21.2**	345.4	206.6	108.7	68.6
QPFMR	52.7	39.7	38.7	42.6	244.5	210.9	208.8	167.6
BRFMR	4223.3	2083.6	1258.2	610.7	4868.9	2224.2	1524.3	769.7

TABLE 5
Varying Number of Objects

Dataset	TripAdvisor-4d				TripAdvisor-7d			
	75K	*100K*	*125K*	*150K*	*75K*	*100K*	*125K*	*150K*
ProbSky	**24.9**	**27.2**	**28.7**	**29.8**	**34.1**	**40.1**	**41.5**	**46.4**
PSky-S	26.2	28.5	30.3	31.5	37.3	42.4	46.4	51.1
PSky-D	26.7	29.1	31.7	36.0	51.6	68.3	86.7	120.1
PSky-R	27.5	29.5	31.8	32.0	39.1	45.9	53.3	61.8
PSky-V	26.6	28.7	33.2	43.7	124.7	206.6	289.7	394.5
QPFMR	34.9	39.7	46.5	73.8	140.6	210.9	249.8	360.0
BRFMR	1590.8	2083.6	4153.8	6103.1	1649.3	2224.2	4732.3	7101.6

TABLE 6
Varying Number of Instances Per Object

Dataset	TripAdvisor-4d				TripAdvisor-7d			
	3	*5*	*7*	*9*	*3*	*5*	*7*	*9*
ProbSky	20.7	**23.8**	**27.2**	**28.1**	**29.0**	**34.1**	**40.1**	**41.7**
PSky-S	20.6	25.3	28.5	29.8	29.4	35.7	42.4	49.4
PSky-D	20.6	24.9	29.1	32.8	29.3	47.0	68.3	99.8
PSky-R	21.3	25.7	29.5	31.8	32.0	38.5	45.9	53.3
PSky-V	**19.6**	24.1	28.7	34.6	42.7	99.9	206.6	334.7
QPFMR	24.2	30.0	39.7	60.6	55.1	129.9	210.9	282.8
BRFMR	735.5	1595.8	2083.6	3820.4	850.8	1510.8	2224.2	4706.2

TABLE 7
Pruning Breakdown When $p = 0.7$

ProbSky	Lemma 1	Lemma 2	DIP
TripAdvisor-4d	244872	1723	7
TripAdvisor-7d	425544	168242	5012
QPFMR	ZPF	UBP	DPF
TripAdvisor-4d	488439	487130	1624
TripAdvisor-7d	646779	623014	88151

TABLE 8
Pruning Breakdown When $p = 0.3$

ProbSky	Lemma 1	Lemma 2	DIP
TripAdvisor-4d	244805	37642	558
TripAdvisor-7d	422127	281330	56336
QPFMR	ZPF	UBP	DPF
TripAdvisor-4d	503664	503559	18144
TripAdvisor-7d	646779	641935	218568

case, PSky-V overshadowed ProbSky and all of its variants. It was because over 99.95% of the instances were eliminated, and thus the number of the remaining instances was too low to gain benefits from any of the optimization techniques. Tables 7 and 8 show the pruning breakdown with two different probability threshold values. Each module shows the number of survival instances after the module was applied. The numbers in the rightmost column are the numbers of candidate instances. A higher number of candidate instances requires more computation resources for the object skyline probability operations. The pruning techniques in PS-QPF-MR and PS-BRF-MR are broken down into three modules: zero-probability filtering (ZPF), upper-bound pruning (UBP), and dominance-power filtering (DPF).

Impact of Number of Machines. Table 4 unveils the performance of all the methods by varying the number of machines from 4 to 32. As shown in Table 4, ProbSky consistently outperformed all the other methods on the TripAdvisor-7d dataset. Initially, the runtimes of all the methods sharply declined with the increasing number of machines until reaching the point of maximum yield; then, the overhead costs started outweighing the parallelization benefits. For example, Table 4 reveals that performance deterioration was observed in PS-QPF-MR when the number of machines exceeded 16 on the TripAdvisor-4d dataset.

Impact of Number of Objects. We studied the impact of cardinality on the performance of all the methods by varying the cardinality from 75K to 150K. Table 5 reveals that increasing cardinality values pushed up the runtimes of all the tested techniques. Compared with PS-QPF-MR and PS-BRF-MR, our method variants' runtimes were less sensitive to the increment of cardinality. In the TripAdvisor-7d case, it was observed that the runtimes were more sensitive than the runtimes in the TripAdvisor-4d case because TripAdvisor-7d had higher dimensionality and resulted in a heavier computational workload with increasing cardinality.

Impact of Number of Instances. We configured the number of instances per object from 3 to 9 to investigate its impact on the performance of all the methods. As presented in Table 6, ProbSky outperformed PS-QPF-MR and PS-BRF-MR in all the configured settings. The performance of ProbSky-B surpassed that of ProbSky only when the number of instances was relatively low. It was because about 0.4% of the instances survived Lemma 2 pruning, and thus the performance of ProbSky was mostly encumbered by the overhead incurred by the communication between machines

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:38:27 UTC from IEEE Xplore. Restrictions apply.

TABLE 9
Varying Number of Dominant Instances

Dataset	TripAdvisor-4d				TripAdvisor-7d			
	2000	*3000*	*4000*	*5000*	*2000*	*3000*	*4000*	*5000*
ProbSky	27.2	39.0	42.1	46.2	40.0	47.0	50.4	53.5

TABLE 10
Varying Threshold Value of Lazy Hyperoctree

Dataset	TripAdvisor-4d				TripAdvisor-7d			
	5	*10*	*15*	*20*	*5*	*10*	*15*	*20*
ProbSky	27.2	32.9	35.1	35.4	40.1	42.7	43.3	43.7

under the MapReduce framework. This issue became neglectable when the number of instances per object or the complexity of the dataset increased. It was observed that the gap between ProbSky and other methods became wider with increasing complexity.

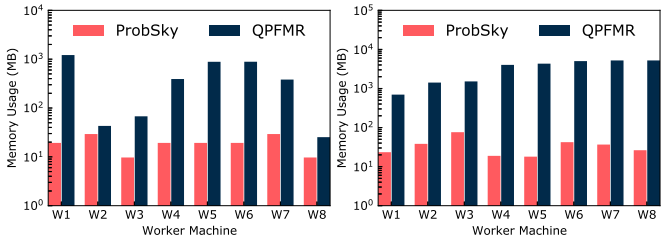
Impact of Number of Dominant Instances. The effect of different numbers of dominant instances on the performance of ProbSky was examined with values varying from 1,000 to 4,000. As shown in Table 9, the performance of ProbSky deteriorated with the increasing number of dominant instances on both datasets.

Impact of Threshold Value of Lazy Hyperoctree. The threshold value for the number of points for a child node to be added was ranged from 5 to 20 to study the effects of lazy hyperoctree with different threshold values on the performance. As shown in Table 10, performance deterioration was observed when the threshold value increased on both TripAdvisor-4d and TripAdvisor-7d datasets.

Impact of Dimensionality. As shown in Tables 3, 4, 5, and 6, ProbSky demonstrated less sensitivity to an increase in dimensionality. It shows our proposed three techniques beneficially reduce the number of dominance tests to a great extent in a high dimensional space.

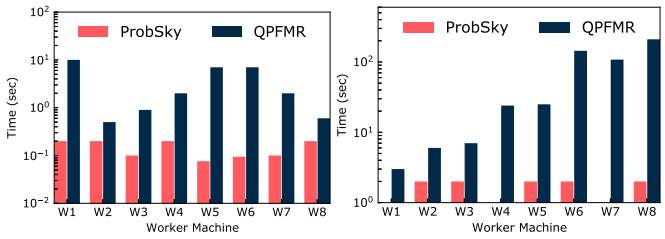
6.2.2 Comparison for Memory Consumption

We leveraged the Spark Web UI [42] to monitor the dynamic execution memory and storage memory consumption used in ProbSky and PS-QPF-MR. The memory consumption reported in this subsection is the summation of the peak execution memory and storage memory. We used eight worker machines in this set of experiments. Fig. 4 shows the memory usage per machine in each experiment. Fig. 4a reveals that the minimum/maximum memory usage among the machines in ProbSky and PS-QPF-MR was 9.7M/29.3M and 25.3M/1.2 G, respectively. Fig. 4b plots ProbSky and PS-QPF-MR's minimum/maximum memory usage among the machines at the rate of 17.9M/42.1M and 671M/5.3G, respectively. On average, ProbSky required 96.0% and 98.9% less total memory usage than PS-QPF-MR on the TripAdvisor-4d and TripAdvisor-7d datasets, respectively.



(a) TripAdvisor 4d Dataset (b) TripAdvisor 7d Dataset

Fig. 4. Comparison of the memory consumption per machine.



(a) TripAdvisor 4d Dataset (b) TripAdvisor 7d Dataset

Fig. 5. Comparison of the execution time per machine.

6.2.3 Comparison for Execution Time per Machine

We recorded the execution time for each worker machine in ProbSky and PS-QPF-MR. The reported execution time starts when the worker began to compute object skyline probabilities after pruning and ends when the worker finished all its assigned tasks. Fig. 5 shows the execution time per machine in ProbSky and PS-QPF-MR. The results plotted in Fig. 5 suggest that the execution time on all the machines in ProbSky was at the same level while the execution time on all the machines in PS-QPF-MR drastically fluctuated.

TABLE 11
Varying Probability Threshold

Dataset	Correlated				Independent				Anti-correlated			
	0.2	0.4	0.6	0.8	0.2	0.4	0.6	0.8	0.2	0.4	0.6	0.8
ProbSky	27.1	26.5	26.1	24.9	**57.9**	**50.6**	**47.0**	**44.6**	**161.2**	**146.8**	**135.0**	**119.9**
PSky-S	25.7	25.0	24.9	24.1	64.0	57.0	51.7	47.0	253.7	229.5	204.2	174.6
PSky-D	26.3	25.6	25.4	24.9	84.0	77.7	68.6	58.5	218.5	212.7	203.5	171.4
PSky-R	26.6	26.4	26.1	25.3	121.3	88.6	73.2	60.0	1062.5	1023.8	803.2	671.2
PSky-V	**24.6**	**24.5**	**24.0**	**23.5**	322.1	299.1	235.3	174.3	2529.1	1928.6	1827.8	1651.6
QPFMR	445.6	403.9	374.1	343.0	1322.6	886.1	619.2	456.1	3445.4	3101.9	2635.2	2113.1

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:38:27 UTC from IEEE Xplore. Restrictions apply.

TABLE 12
Varying Number of Machines

Dataset	Correlated				Independent				Anti-correlated			
	4	*8*	*16*	*32*	*4*	*8*	*16*	*32*	*4*	*8*	*16*	*32*
ProbSky	27.7	26.1	20.6	21.6	**53.7**	**47.0**	**35.4**	**34.1**	**169.0**	**135.0**	**93.8**	**80.3**
PSky-S	26.9	24.9	20.2	**20.0**	56.2	51.7	38.4	34.7	244.5	204.2	132.0	95.5
PSky-D	27.1	25.4	21.1	21.1	84.9	68.6	52.2	41.5	242.2	203.5	130.2	104.2
PSky-R	27.8	26.1	20.4	20.5	98.7	73.2	47.9	40.1	1715.7	803.2	580.0	353.2
PSky-V	**25.0**	**24.0**	**19.3**	20.0	393.5	235.3	150.6	97.3	4116.7	1827.8	986.7	645.2
QPFMR	915.0	374.1	334.4	326.4	827.8	619.2	571.6	524.6	4371.5	2635.2	2462.2	2662.2

TABLE 13
Varying Number of Objects

Dataset	Correlated				Independent				Anti-correlated			
	50K	*100K*	*200K*	*400K*	*50K*	*100K*	*200K*	*400K*	*50K*	*100K*	*200K*	*400K*
ProbSky	20.5	22.6	26.1	32.5	33.3	**40.6**	**47.0**	**63.3**	52.2	**79.4**	**135.0**	**241.4**
PSky-S	19.3	21.7	24.9	31.9	32.7	41.8	51.7	70.4	56.3	103.7	204.2	451.1
PSky-D	20.1	22.9	25.4	29.1	**29.8**	44.7	68.6	143.1	**50.8**	88.7	203.5	583.1
PSky-R	21.0	21.3	26.1	33.5	37.6	53.6	73.2	146.4	152.1	303.3	803.2	3408.2
PSky-V	**18.2**	**20.1**	**24.0**	**28.5**	47.2	102.0	235.3	760.8	237.5	526.8	1827.8	5987.2
QPFMR	88.0	172.5	374.1	845.2	68.7	156.5	619.2	1893.8	169.9	657.3	2635.2	-

TABLE 14
Varying Number of Instances Per Object

Dataset	Correlated				Independent				Anti-correlated			
	5	*10*	*15*	*20*	*5*	*10*	*15*	*20*	*5*	*10*	*15*	*20*
ProbSky	23.6	26.1	33.1	33.2	**37.7**	**47.0**	**60.3**	**69.6**	**75.8**	**135.0**	**220.9**	**327.3**
PSky-S	21.6	24.9	33.1	33.4	39.7	51.7	69.0	82.1	93.6	204.2	396.1	650.5
PSky-D	22.3	25.4	**29.5**	**30.7**	38.0	68.6	126.5	204.9	78.9	203.5	428.2	656.5
PSky-R	23.7	26.1	33.9	34.8	45.4	73.2	117.8	219.2	250.8	803.2	3167.6	4814.4
PSky-V	**21.2**	**24.0**	31.2	31.7	59.6	235.3	629.4	1375.5	426.5	1827.8	5012.6	-
QPFMR	148.4	374.1	507.3	941.4	150.6	619.2	1810.6	3690.4	670.2	2635.2	3953.9	9301.7

TABLE 15
Varying Number of Dominant Instances

Dataset	Correlated				Independent				Anti-correlated			
	2000	*3000*	*4000*	*5000*	*2000*	*3000*	*4000*	*5000*	*2000*	*3000*	*4000*	*5000*
ProbSky	26.3	25.9	25.7	26.1	44.0	45.5	46.7	46.9	152.9	149.0	145.4	135.0

6.3 Performance Comparison on Synthetic Data

To validate the superiority of ProbSky under different data distributions, we further employed a benchmark skyline dataset generator [2], [37] to originate three synthetic datasets (correlated, independent, and anti-correlated). In these three datasets, the domain of each dimension was [0, 1) and the edge size of each hyper-rectangle was configured at 0.05. All the instances of an object were uniformly generated in each hyper-rectangle. By default, the cardinality was 200K, the number of instances per object was 10, the number of machines was 8, the probability threshold was 0.6, the number of dominant instances was 5,000, and the threshold value of the lazy hyperoctree was 5. It is worth noting that PS-BRF-MR could not finish within 10^4 seconds, and is not presented in the following tables. Runtimes are marked (-) when a timeout of 10^4 seconds was reached. Bold text indicates the method with the lowest runtime for each setting.

6.3.1 Comparison for Efficiency

Impact of Probability Threshold. Table 11 unveils the performance of all the schemes by varying the probability threshold under three different distributions. The results depicted in Table 11 signify that ProbSky consistently outperformed all the other competitors under the independent and anti-correlated distributions and had a slight disadvantage against its variants under the correlated distribution. It was because over 99.4% of the instances were pruned (when $p \geq 0.2$) before any of the optimization techniques were applied. The overhead incurred

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:38:27 UTC from IEEE Xplore. Restrictions apply.

TABLE 16
Varying Threshold Value of Lazy Hyperoctree

Dataset	Correlated				Independent				Anti-correlated			
	5	*10*	*15*	*20*	*5*	*10*	*15*	*20*	*5*	*10*	*15*	*20*
ProbSky	26.1	26.4	26.6	26.8	46.9	47.8	49.3	50.8	135.0	146.8	163.3	184.4

by the added optimization techniques under the MapReduce framework was heavier than direct computation. The effectiveness of the pruning techniques under various distributions was also empirically evaluated. Tables 17 and 18 show the number of instances left after each technique was applied when the probability threshold was configured as 0.7 and 0.3, respectively.

Impact of Number of Machines. Table 12 depicts the effect of the number of machines on the performance of all the methods on the three synthetic datasets. As depicted in Table 12, ProbSky consistently outperformed all other methods under the independent and anti-correlated distributions. The runtime measures of all the methods plunged initially, followed by a dwindling slope with an increase in the number of machines. The largest performance gain of ProbSky over baseline methods can be observed on the independent and anti-correlated datasets.

Impact of Number of Objects. Table 13 depicts the impact of cardinality on the performance of all the methods by varying the cardinality from 50K to 400K on the three synthetic datasets. As depicted in Table 13, ProbSky had an inevitable disadvantage against its variants under the correlated distribution. It was because the instances under the correlated distribution were mostly pruned prior to the three optimization techniques, impeded by the overhead incurred from multiple MapReduce passes when the number of machines increased. Under the independent and anti-correlated distributions, although ProbSky initially performed slightly worse than ProbSky-D when the number of objects was low, it exhibited superior performance against all the other methods when the number of objects increased. The performance gain became more noticeable as the number of objects increased.

Impact of Number of Instances. Table 14 quantifies the impact of the number of instances per object on the performance of all the methods by varying the number of instances from 5 to 20 on the three synthetic datasets. Table 14 shows that ProbSky consistently yielded a much shorter runtime than those of all the other methods under the independent and anti-correlated distributions.

Impact of Number of Dominant Instances. Table 15 presents the performance of ProbSky with the increasing number of dominant instances under different data distributions. Under the correlated distribution, the performance initially increased when the number of the dominant instances increased and then started to deteriorate. This was because the increase in the number of dominant instances did not effectively contribute to pruning after a certain value, and thus it suffered from the overhead incurred and led to a decline in the performance. Under the independent distribution, the performance declined along with the increasing number of dominant instances. By contrast, the performance increased with the increasing number of dominant instances under the anti-correlated distribution.

Impact of Threshold Value of Lazy Hyperoctree. Table 16 shows the performance of ProbSky under three different data distributions using different threshold values. As presented in Table 16, the performance deteriorated with an increase in the number of points for a child node to be added.

TABLE 17
Pruning Breakdown When $p = 0.7$

ProbSky	Lemma 1	Lemma 2	DIP
Correlated	28397	8026	938
Independent	432161	238697	30152
Anticorrelated	1142597	995864	572749
QPFMR	ZPF	UBP	DPF
Correlated	1762843	1617793	28440
Independent	1983506	1979943	324747
Anticorrelated	2000000	1997590	964650

TABLE 18
Pruning Breakdown When $p = 0.3$

ProbSky	Lemma 1	Lemma 2	DIP
Correlated	28922	16708	4013
Independent	436447	364325	87513
Anticorrelated	1170636	1130976	853642
QPFMR	ZPF	UBP	DPF
Correlated	1760244	1721499	57252
Independent	1991040	1986390	528460
Anticorrelated	2000000	1999730	1417440

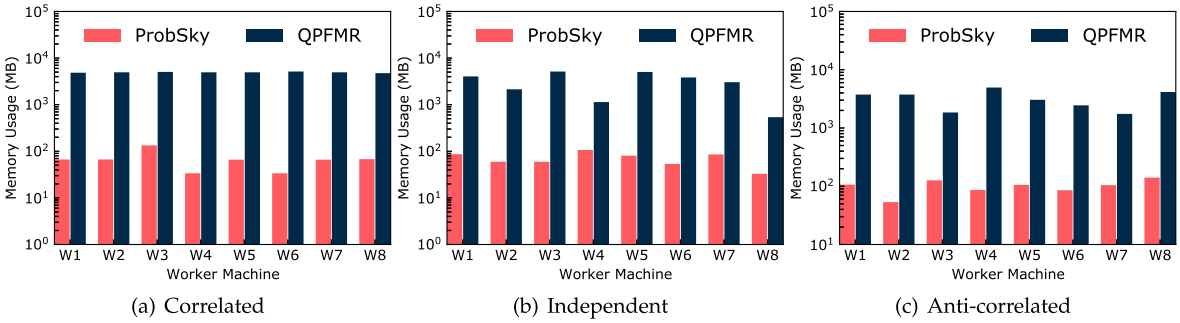
6.3.2 Comparison for Memory Consumption

Fig. 6 shows the memory usage for each worker machine in ProbSky and PS-QPF-MR under correlated, independent, and anti-correlated distributions. As depicted in Fig. 6, ProbSky required substantially less memory space under all the settings than PS-QPF-MR. Note that a few worker machines reached the maximum memory usage in PS-QPF-MR, resulting in shuffle spills. On average, ProbSky reduced by 98.6%, 97.7%, and 96.8% total memory usage compared to PS-QPF-MR on the correlated, independent, and anti-correlated datasets, respectively. In addition, some of the machines exceeded the maximum memory they can hold when running PS-QPF-MR and resulted in spill overhead of 33.2G, 11.1G, 5.4G on the correlated, independent, and anti-correlated datasets, respectively.

6.3.3 Comparison for Execution Time per Machine

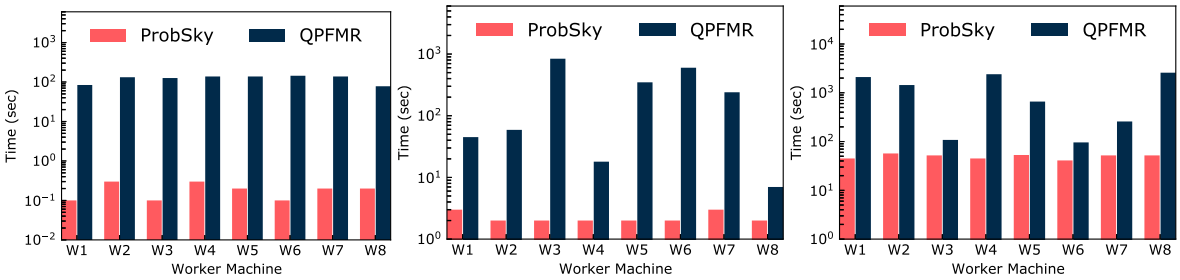
Fig. 7 plots the execution time for each worker machine in ProbSky and PS-QPF-MR under the correlated, independent, and anti-correlated distributions. Fig. 7 reveals that the severe bottlenecks and runtime fluctuations were observed in PS-QPF-MR due to skewed workloads. We noticed from the experimental results that a few worker nodes took an

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:38:27 UTC from IEEE Xplore. Restrictions apply.



(a) Correlated (b) Independent (c) Anti-correlated

Fig. 6. Comparison of the memory consumption per machine.



(a) Correlated (b) Independent (c) Anti-correlated

Fig. 7. Comparison of the execution time per machine.

extremely long time to complete whereas some of the other worker nodes finished computing within only a few seconds. In contrast, ProbSky successfully obtained a much more even distribution on the execution time for all the machines than PS-QPF-MR thanks to our proposed load-balancing mechanism.

7 Conclusion

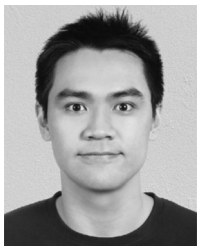
In this research, a novel MapReduce-based framework was implemented, *ProbSky*, in support of fast parallel and distributed evaluation of the probabilistic skyline queries on large high-dimensional data. ProbSky is adroit at efficiently evaluating the exact p-skyline queries on large uncertain data without compromising the quality of query results. ProbSky embraces three optimization techniques: dominant instance pruning, slab-based partitioning, and reference point-based acceleration. These extensive experiments driven by both real and synthetic datasets, demonstrate that compared to other state-of-the-art methodologies, ProbSky is conducive to speeding up the evaluation of the exact p-skyline queries on large high-dimensional data by at least one order of magnitude in most cases. Additional experiments also validate that by balancing the memory consumption and execution time among machines, ProbSky alleviates the bottleneck problem causing severe system performance deterioration. The source code can be downloaded at http://github.com/KuoAite/ProbSky.

References

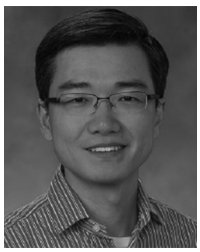
[1] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting," in *Proc. 19th Int. Conf. Data Eng.*, 2003, pp. 717–719.
[2] S. Börzsönyi, D. Kossmann, and K. Stocker, "The skyline operator," in *Proc. IEEE 19th Int. Conf. Data Eng.*, 2001, pp. 421–430.
[3] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: An online algorithm for skyline queries," in *Proc. 28th Int. Conf. Very Large Databases*, 2002, pp. 275–286.
[4] K. C. K. Lee, B. Zheng, H. Li, and W.-C. Lee, "Approaching the skyline in z order," in *Proc. 28th Int. Conf. Very Large Databases*, 2007, pp. 279–290.
[5] M. Hua, J. Pei, W. Zhang, and X. Lin, "Ranking queries on uncertain data: A probabilistic threshold approach," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 673–686.
[6] J. Li, B. Wang, and G. Wang, "Efficient probabilistic reverse K-nearest neighbors query processing on uncertain data," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2013, pp. 456–471.
[7] F. Li, K. Yi, and W. Le, "Top-k queries on temporal data," *Very Large Data Base J.*, vol. 19, no. 5, pp. 715–733, 2010.
[8] J. Pei, B. Jiang, X. Lin, and Y. Yuan, "Probabilistic skylines on uncertain data," in *Proc. 33rd Int. Conf. Very Large Data Bases*, 2007, pp. 15–26.
[9] M. J. Atallah and Y. Qi, "Computing all skyline probabilities for uncertain data," in *Proc. 28th ACM SIGMOD-SIGACT-SIGART Symp. Princ. Database Syst.*, 2009, pp. 279–287.
[10] C. Böhm, F. Fiedler, A. Oswald, C. Plant, and B. Wackersreuther, "Probabilistic skyline queries," in *Proc. 18th ACM Conf. Inf. Knowl. Manage.*, 2009, pp. 651–660.
[11] D. Kim, H. Im, and S. Park, "Computing exact skyline probabilities for uncertain databases," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 12, pp. 2113–2126, Dec. 2012.
[12] F. N. Afrati, P. Koutris, D. Suciu, and J. D. Ullman, "Parallel skyline queries," in *Proc. 15th Int. Conf. Database Theory*, 2012, pp. 274–284.
[13] S. Park, T. Kim, J. Park, J. Kim, and H. Im, "Parallel skyline computation on multicore architectures," in *Proc. IEEE 25th Int. Conf. Data Eng.*, 2009, pp. 760–771.
[14] P. Wu, C. Zhang, Y. Feng, B. Y. Zhao, D. Agrawal, and A. El Abbadi, "Parallelizing skyline queries for scalable distribution," in *Proc. Int. Conf. Extending Database Technol.*, 2006, pp. 112–130.
[15] A. Cosgaya-Lozano, A. Rau-Chaplin, and N. Zeh, "Parallel computation of skyline queries," in *Proc. 21st Int. Symp. High Perform. Comput. Syst. Appl.*, 2007, Art. no. 12.
[16] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.
[17] W. Lu, Y. Shen, S. Chen, and B. C. Ooi, "Efficient processing of K nearest neighbor joins using mapreduce," *Proc. VLDB Endowment*, vol. 5, no. 10, pp. 1016–1027, Jun. 2012.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:38:27 UTC from IEEE Xplore. Restrictions apply.

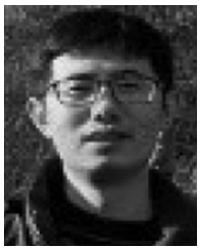
[18] C. Zhang, F. Li, and J. Jestes, "Efficient parallel kNN joins for large data in mapreduce," in *Proc. 15th Int. Conf. Extending Database Technol.*, 2012, pp. 38–49.
[19] R. Vernica, M. J. Carey, and C. Li, "Efficient parallel set-similarity joins using mapreduce," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2010, pp. 495–506.
[20] A. Okcan and M. Riedewald, "Processing theta-joins using mapreduce," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 949–960.
[21] K.-L. Tan, P.-K. Eng, and B. C. Ooi, "Efficient progressive skyline computation," in *Proc. 27th Int. Conf. Very Large Data Bases*, 2001, pp. 301–310.
[22] M. D. Morse, J. M. Patel, and H. V. Jagadish, "Efficient skyline computation over low-cardinality domains," in *Proc. 33rd Int. Conf. Very Large Data Bases*, 2007, pp. 267–278.
[23] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 41–82, 2005.
[24] S. Zhang, N. Mamoulis, and D. W. Cheung, "Scalable skyline computation using object-based space partitioning," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2009, pp. 483–494.
[25] K. C. K. Lee, W. Lee, B. Zheng, H. Li, and Y. Tian, "Z-SKY: An efficient skyline query processing framework based on Z-order," *VLDB J.*, vol. 19, no. 3, pp. 333–362, 2010.
[26] J. Zhang, W. Wang, X. Jiang, W. Ku, and H. Lu, "An MBR-oriented approach for efficient skyline query processing," in *Proc. IEEE 35th Int. Conf. Data Eng.*, 2019, pp. 806–817.
[27] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.
[28] B. Zhang, S. Zhou, and J. Guan, "Adapting skyline computation to the mapreduce framework: Algorithms and experiments," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2011, pp. 403–414.
[29] A. Vlachou, C. Doulkeridis, and Y. Kotidis, "Angle-based space partitioning for efficient parallel skyline computation," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 227–238.
[30] L. Chen, K. Hwang, and J. Wu, "MapReduce skyline query processing with a new angular partitioning approach," in *Proc. IEEE 26th Int. Parallel Distrib. Process. Symp. Workshops PhD Forum*, 2012, pp. 2262–2270.
[31] J. Zhang, X. Jiang, W. Ku, and X. Qin, "Efficient parallel skyline evaluation using MapReduce," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 7, pp. 1996–2009, Jul. 2016.
[32] H. Köhler, J. Yang, and X. Zhou, "Efficient parallel skyline processing using hyperplane projections," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 85–96.
[33] H. Wijayanto, W. Wang, W.-S. Ku, and A. Chen, "LShape partitioning: Parallel skyline query processing using mapreduce," *Proc. IEEE 37th Int. Conf. Data Eng.*, 2021, pp. 2340–2341.
[34] K. Mehlhorn, *Data Structures and Algorithms 3: Multi-Dimensional Searching and Computational Geometry*. New York; Berlin: Springer, 1984.
[35] X. Lian and L. Chen, "Reverse skyline search in uncertain databases," *ACM Trans. Database Syst.*, vol. 35, no. 1, pp. 1–49, Feb. 2008.
[36] L. Ding, G. Wang, J. Xin, and Y. Yuan, "Efficient probabilistic skyline query processing in mapreduce," in *Proc. IEEE Int. Congr. Big Data*, 2013, pp. 203–210.
[37] Y. Park, J.-K. Min, and K. Shim, "Processing of probabilistic skyline queries using mapreduce," *Proc. VLDB Endowment*, vol. 8, no. 12, pp. 1406–1417, Aug. 2015.
[38] K. Mullesgaard, J. L. Pederseny, H. Lu, and Y. Zhou, "Efficient skyline computation in MapReduce," in *Proc. 26th Int. Conf. Extending Database Technol.*, 2014, pp. 37–48.
[39] E. Gavagsaz, "Parallel computation of probabilistic skyline queries using mapreduce," *J. Supercomputing*, vol. 77, pp. 418–444, 2021.
[40] R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys," *Acta Inf.*, vol. 4, no. 1, pp. 1–9, Mar. 1974.
[41] D. Antognini and B. Faltings, "Hotelrec: A novel very large-scale hotel recommendation dataset," in *Proc. 12th Lang. Resour. Eval. Conf.*, 2020, pp. 4917–4923.
[42] *Spark Web UI*, Apache Spark, Nov. 2021. [Online]. Available: https://spark.apache.org/docs/latest/



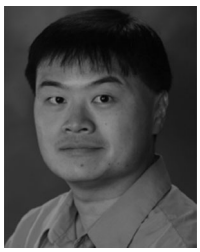
Ai-Te Kuo received the bachelor's degree in computer science and information engineering from National Taipei University, in 2015 and the MS degree in computer science and software engineering from Auburn University, in 2019. He is currently working toward the PhD with Auburn University. His research interests include data mining and cloud security.



Haiquan Chen received the ME degree and the BE degree in computer science from Xian Jiaotong University (XJTU) in China, in 2006 and 2003, respectively and the PhD degree in computer science from Auburn University, in 2011. He is an assistant professor with the Department of Computer Science with California State University, Sacramento. He was an assistant/associate professor with the Department of Computer Science with Valdosta State University, Valdosta, GA from 2011 to 2017. His research interests embrace database and Big Data management, with emphasis on data mining, machine learning, location-based services, and social network analysis. He is a member of the ACM and the ACM SIGMOD.



Liang Tang received the BS degree in computer science from Xi'an Jiaotong University, in 2011 and the PhD degree in computer science from Auburn University, in 2016, under the supervision of Dr. Wei-Shinn Ku and Dr. Xiao Qin. He is a software engineer in data field with Facebook. Before joining Facebook in 2020, he worked as a software engineer with Linkedin.



Wei-Shinn Ku (Senior Member, IEEE) received the MS degree in computer science and the MS degree in electrical engineering from USC, in 2003 and 2006, respectively and the PhD degree in computer science from the University of Southern California (USC), in 2007. He is a program director with the National Science Foundation and a professor with the Department of Computer Science and Software Engineering with Auburn University. His research interests include databases, data science, mobile computing, and cybersecurity. He has published more than 140 research papers in refereed international journals and conference proceedings. He is a member of the ACM SIGSPATIAL.



Xiao Qin (Senior Member, IEEE) received the BS and MS degrees in computer science from the Huazhong University of Science and Technology, in 1992 and 1999, respectively and the PhD degree in computer science from the University of Nebraska-Lincoln, in 2004. He is currently an Alumni professor and Director of Graduate Programs in the Department of Computer Science and Software Engineering with Auburn University. Prior to joining Auburn University, in 2007, he had been an assistant professor with New Mexico Institute of Mining and Technology (New Mexico Tech) for three years. He won an NSF CAREER Award in 2009. His research is supported by the US National Science Foundation (NSF). His research interests include parallel and distributed systems, storage systems, fault tolerance, real-time systems, and performance evaluation.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:38:27 UTC from IEEE Xplore. Restrictions apply.