Efficient Parallel Skyline Evaluation Using MapReduce

Ji Zhang, *Member, IEEE*, Xunfei Jiang, *Member, IEEE*, Wei-Shinn Ku, *Senior Member, IEEE*, and Xiao Qin, *Senior Member, IEEE*

Abstract—This research develops an advanced two-phase MapReduce solution that is able to efficiently address skyline queries on large datasets. Unlike existing parallel skyline approaches, our scheme considers data partitioning, filtering, and parallel skyline evaluation as a holistic query process. In particular, we apply filtering techniques and angle-based partitioning in the first phase, in which unqualified objects are discarded and the processed objects are partitioned by their angles to the origin.In the second phase, local skyline objects in each partition are calculated in parallel, and global skyline objects are output after a merging skyline process. To improve the parallel local skyline calculation, we propose two partition-aware filtering methods that keep skyline candidates in a balanced manner. The aggressive partition-aware filtering aggressively eliminates objects in the partition with the greatest population of candidate objects, whereas the proportional partition-aware filtering slows down the growth of partition population proportionally. Recognizing the lack of studies that incorporate the MapReduce framework into parallel skyline processing, we propose a partial-presort grid-based partition skyline algorithm that is able to significantly improve the merging skyline computation on large datasets. The presort process can be completed in the shuffle phase with little overhead. Our experimental results show the efficiency and effectiveness of the proposed parallel skyline solution utilizing MapReduce on large-scale datasets.

Index Terms—Skyline query, mapreduce, big data

1 Introduction

Since being introduced into database research, the skyline operator has become increasingly important in multi-criteria decision-making applications [7], [18]. Given a set of multidimensional objects, the skyline query returns an object set, in which each object is not dominated by any other objects. Fig. 1 demonstrates an example that contains hotels with their price and the distance from the beach. Hotels {a, e, h, i, j} are the skyline objects because there is not any hotel that is either lower in price (with the same distance), shorter in distance (with the same price), or better in both dimensions than them.

A number of efficient algorithms have been proposed for the skyline evaluation. For example, Bitmap [32], Index [32], Nearest Neighbor (NN) [21], Branch-and-Bound Skyline (BBS) [27], Lattice Skyline (LS) [25], and ZSearch [23] rely on indices, which are assumed to be constructed before query processing. On the other hand, non-index approaches include Block-Nested-Loop (BNL) [7], Divide and Conquer (D&C) [7], Sort-Filter-Skyline (SFS) [10], Linear Elimination Sort for Skyline (LESS) [17], Sort and Limit Skyline (SaLSa) [5], VSkyline [9], and Object-based Space Partitioning Skyline (OSPS) [38]. Their experimental results showed that the methods using indexing techniques are more efficient, since they can access data by indices instead of scanning an entire dataset.

Due to high computation complexity of the skyline evaluation on large datasets, parallel skyline approaches have been studied based on the skyline additivity property. A dataset can be divided into a number of subsets. For each subset, a local skyline evaluation is executed. Then, merging skyline computation is invoked for a final check [34], [35], [11], [26].

However, the following problems make existing parallel solutions impractical. First, a large amount of data is dynamically produced. For example, one million transactions are made per hour in Walmart stores [2], and one billion photos are uploaded by Facebook users every day [6]. The cost of dynamically constructing indices or data partitions is considerably high. Second, attribute values of objects keep changing. If the indices and data partitions are created at a preprocessing stage, maintaining them is prohibitively expensive. For instance, 2.7 billion Facebook "likes" and comments are generated per day [15], which would make a rarely visited restaurant be popular in one day. Moreover, the effort of maintaining indices in an incremental manner will be unnecessary, if skyline operations are not frequently performed. Third, none of the existing parallel solutions has investigated the efficiency of data partitioning, skyline filtering, and parallel skyline computation as a holistic skyline query evaluation framework. Fourth, the MapReduce framework [13] has been incorporated into skyline computation [36], [8] and an increasing number of database applications [24], [37]. Two skyline solutions that apply existing skyline algorithms in the MapReduce framework have been presented [8], [36]. However, methods that presort objects in the shuffle phase, which potentially reduces the presorting cost and skyline calculation, have not been investigated.

- *J. Zhang, W.-S. Ku, and X. Qin are with the Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849. E-mail: {jizhang, weishinn, xqin}@auburn.edu.*
- *X. Jiang is with the Department of Computer Science, Earlham College, Richmond, IN 47374. E-mail: jiangxu@earlham.edu.*

Manuscript received 11 Oct. 2014; revised 15 Aug. 2015; accepted 18 Aug. 2015. Date of publication 23 Aug. 2015; date of current version 15 June 2016. Recommended for acceptance by Y. Wang.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2015.2472016

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:33:12 UTC from IEEE Xplore. Restrictions apply.
1045-9219 © 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

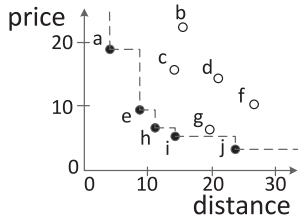


Fig. 1. A skyline example.

We propose a novel two-phase MapReduce-based solution in this paper. In the first phase, the input dataset is evenly partitioned into multiple subsets by a random data partitioning. For each subset, a map function applies filtering techniques that filter out unqualified objects and partitions the processed objects by angle-based partitioning. The reduce function further filters out objects after aggregating them from map functions. In the second phase, the map function calculates local skylines in each angle-based partition. The reduce function merges intermediate results and produces global skyline objects. Compared to one-phase MapReduce-based solutions [8], [36], our solution suffers from overheads of one more phase processing, however the MapReduce framework and the filtering methods in our solution are efficient. Since the execution time and I/O cost of skyline calculation will grow dramatically when the input dataset becomes large, applying the filtering methods twice could eliminate more objects before skyline calculation and save more time in the entire processing. The angle-based partitioning is selected in the first phase because it could filter out more objects than grid-based partitioning [33].

In addition, due to unawareness of the distribution of objects after skyline filtering, existing filtering methods, such as MaxSum and MaxDist [12], produce much fewer objects in central partitions than others. If each partition will be processed in parallel in the second phase, processing the *fattest partition* with the greatest number of objects will be the bottleneck, because the execution time of a parallel program is determined by the slowest process, and skyline computation takes longer on larger inputs. In order to alleviate the imbalance, we propose two partition-aware skyline filtering algorithms, which aggressively or proportionally reduce the object population of the fattest partition, respectively.

In the second phase, a large number of local skyline objects are merged by a reduce function. Simply using an existing non-index skyline algorithm in the reduce function suffers from inefficient I/O accesses. In addition, the reduce function cannot be performed in parallel in our solution. Thus, we proposed an efficient Partial-presort Grid-based Partition Skyline (PGPS) algorithm, which aims to reduce object comparison and the number of passes by assuming that the input objects have been sorted in a particular order (i.e., objects cannot be dominated by the ones in subsequent partitions). The presorting can be easily incorporated into the MapReduce framework.

We make the following contributions in this paper.

- We propose a parallel framework to efficiently evaluate skyline queries on a large dataset using MapReduce.
- We design an aggressive and a proportional partition-aware filtering (PPF) methods, which keep objects among partitions in a balanced manner.
- We design a novel PGPS algorithm that can efficiently answer skyline queries without indices. The algorithm can also be easily incorporated into the MapReduce framework.
- We evaluate the performance of the proposed solutions through extensive experiments with large-scale datasets.

The rest of this paper is organized as follows. Section 2 surveys related work. The skyline query and relevant techniques utilized in our solutions are formally defined in Section 3. Our advanced two-phase MapReduce-based solution is illustrated in Section 4. The experimental validation of our design is presented in Section 5. Section 6 concludes the paper.

2 Related Work

The skyline query has been extensively studied in the literature. The skyline operator was originally studied in the maximal vector problem [22] and first introduced in database management systems by [7]. In their studies, a number of external memory algorithms were proposed, such as BNL and D&C. The BNL algorithm uses a block nested loop to compare each object with every other object in the dataset. A skyline object cannot be dominated by any other object. In the D&C algorithm, the dataset is recursively partitioned into smaller subsets. Each subset can be easily loaded into memory for local skyline computation. Then, all local skyline objects are merged by a merging skyline approach. SFS [10] and LESS [17] reduce the number of data scans by sorting objects according to their values in a particular dimension. Because no object can be dominated by subsequent objects, object replacement is not required in these methods. Another sort-based algorithm, SaLSa, was designed in [5], in which sorting objects is used as a method to stop retrieving data from the input stream. Cho et al. designed VSkyline that optimizes dominance tests by employing vectorization in SIMD (single instruction, multiple data) architectures [9].

Tan et al. proposed two progressive algorithms, Bitmap and Index [32]. The Bitmap algorithm abstracts objects as bit vectors and uses bit-wise operations on object comparison. The Index algorithm exploits the data transformation mechanism and the B^+-tree index to answer skyline query. Kossmann et al. designed a Nearest Neighbor approach that recursively applies NN search over different data regions [21]. Papadias et al. proposed a Branch-and-Bound Skyline method based on the best-first nearest neighbor algorithm [27]. Motivated by a close connection between the Z-order curve and skyline processing strategies, Lee et al. proposed a new index structure, ZBtree, that stores objects based on the Z-order curve [23]. Moreover, ZSearch was developed to progressively produce skyline objects; ZUpdate supports incremental skyline result maintenance, and k-ZSearch can answer k-dominant skyline query. Morse et al. focused on skyline queries in low-cardinality domains and designed Lattice Skyline algorithm that can achieve linear computation complexity to the number of input objects [25]. Zhang et al. developed a dynamic indexing technique that can be easily integrated into sort-based skyline algorithms [38].

Balke et al. addressed skyline queries in distributed environments [4]. In their method, object attributes are vertically

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:33:12 UTC from IEEE Xplore. Restrictions apply.

partitioned in such a manner that each partition records object attributes in one dimension. The method first calculates skyline objects in parallel and reports them to a central point. Then, all non-skyline objects are removed from those objects with at least one dimension value retrieved. Wu et al. proposed a parallel skyline query evaluation over large-scale distributed environments by leveraging on content-based data partitioning [35]. By utilizing recursive region partitioning and dynamic region encoding mechanisms, their method can not only avoid unnecessary data access in the regions that do not have potential skyline objects, but it can also progressively report results. Their method provides incremental scalability, in which workload can be automatically balanced by distributing objects to new nodes.

Wang et al. investigated skyline evaluation on a balanced tree structured peer-to-peer network [34]. They designed an adaptive skyline space partitioning approach based on query accessing patterns, in which load balancing can be achieved when a node joins or leaves the network. By accurately estimating the nodes within particular sub-spaces, the method can reduce the complexity of skyline computation by controlling the amount of query forwarding, limiting the number of peers and message transmission over network. Huang et al. proposed techniques to reduce the execution time and the costs of communication among mobile devices on a mobile network [19]. Because each mobile device is capable of holding a portion of datasets in a network, queries have to be forwarded through the entire network. They proposed a filtering method that can significantly reduce the data transferred among devices. Cui et al. proposed a novel algorithm PaDSkyline (Parallel Distributed Skyline query processing) for large-scale distributed environments [12]. Their method considerably shortens the response time by performing skyline evaluation in parallel. Before the skyline processing, filtering points are set up to filter out unqualified objects, which is able to reduce the size of datasets processed by skyline computation. Gao et al. investigated the skyline queries in multiple disk environments [16]. In their method, objects are distributed among multiple disks and accessed in parallel by using parallel R-trees. Park et al. evaluated two parallel skyline approaches on multi-core architectures [28]. The first method parallelized BBS algorithm, in which a special index is used to eliminate dominance tests. The second method, pskyline, executes D&C method in parallel by using skeletal parallel programming.

A random data partitioning method is applied to the parallel skyline computation in order to ensure that object distribution in each partition is similar to the original dataset [11]. In grid-based partitioning methods, datasets are divided into smaller subsets, each of which can be processed on servers in parallel [3], [30], [34], [35]. Vlachou et al. proposed an angle-based data partitioning method, which maps objects from the Cartesian coordinate space into a hypershperical space [33]. Objects are partitioned by their angular coordinates. The advantage of this method is that the average pruning power of objects can be increased within a partition, and the number of skyline objects in local skyline computation can be minimized. Due to high computation complexity of coordinate conversion, Köhler et al. partitioned datasets by using hyperplane projections [20]. Their method can achieve small local skyline sets after the partitioning of data and efficient merging of results.

Park et al. investigated skyline and reverse skyline queries using MapReduce [29]. Their method first partitions the data space by using a quad-tree, which can be created by randomly selecting samples from the dataset. Then, the skyline query is evaluated in local skyline and global skyline phases. The performance of their method largely relies on the data distribution under quad-tree partitioning and sample selection. However, our approach assumes that the dataset is dynamically generated. Selecting a small set of samples that fully represent the characteristics of a dynamic dataset is non-trivial; accessing a large portion of the dataset is expensive. Eldawy et al. proposed CG_Hadoop, a suite of MapReduce algorithms, to solve fundamental computational geometry problems [14]. They extended existing skyline solutions by applying the Hadoop platform and utilizing the R-tree index to improve performance on the SpatialHadoop platform. Zhang et al. incorporated BNL, SFS, and Bitmap approaches into MapReduce [36]. The MapReduce-BNL method partitions datasets into smaller disjoint subsets by using a grid-based partitioning method. Each map function calculates local skylines on a subset and a single reduce function merges the intermediate results and outputs the global skylines. The MapReduce-SFS method follows the same process, but applies the presorting technique in local skyline calculation instead of using BNL. The MapReduce-Bitmap method leverages the bitmap algorithm [32] in skyline calculation. Chen et al. utilized an angular data partition in their MapReduce Skyline query processing [8]. In their MapReduce-Angle method, each mapper calculates local skylines on an angle-based partition, and a single reducer completes the global skyline determination. Mullesgaard et al. proposed a parallel skyline solution in MapReduce, which uses bit strings to filter out unqualified skyline candidates and generates independent partition groups for merging skyline computation in parallel [26].

Our proposed solution differs from the aforementioned studies in the following three aspects. First, we focus on answering skyline queries on large amounts of data in parallel. We consider data partitioning, skyline filtering, and parallel skyline calculation as an entire skyline process. Second, our approach receives raw data as input, so that we do not use any indexing and pre-computation techniques. Creating indices at runtime is very expensive. Third, we propose a novel MapReduce-based skyline solution, in which two partition-aware filtering methods and a Partial-presort Grid-based Partition Skyline algorithm are incorporated. With the advantage of the grid-based partial order, PGPS is able to reduce the number of iterations and I/O operations suffered by BNL, and avoid an expensive external sorting required by SFS. More importantly, PGPS can be easily incorporated in the shuffle phase of the MapReduce framework with minor overhead.

3 Preliminaries

3.1 Problem Statement

Formally, given a dataset P in a d-dimensional data space, an object $p \in P$ can be represented as $p = \{x_1, x_2, \ldots, x_d\}$ where $p.x_i$ is the value of the object on the ith dimension.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:33:12 UTC from IEEE Xplore. Restrictions apply.

TABLE 1
Symbolic Notations

Symbol	Meaning
p	an object
$p.x_i$	the value of the object p in the ith dimension in Cartesian coordinate space
$p.\phi_i$	the angle of the object p in the ith dimension in hyperspherical coordinate space
P	a dataset
P_i	the ith partition (a subset of a dataset)
d	data dimensionality
R_i	a range value in the ith dimension
V	a partition
l_i	lower bounds in the ith dimension
u_i	upper bounds in the ith dimension
$p \prec q$	object p dominates object q
V^A	angle-based partition
V^G	grid-based partition
$\|V\|$	the number of partitions
$SKY(P)$	skyline set of dataset P

Without loss of generality, we assume that the minimum values in all dimensions are preferred in this paper. Notations used in this paper are summarized in Table 1. The domination relationship $\prec$ and the skyline operator, denoted by SKY(P), are defined as follows [3], [18]:

Definition (Domination). *Given two objects p, $q \in P$, p dominates q, denoted as $p \prec q$, if $\forall i \in \{1, \ldots, d\},\ p.x_i \leq q.x_i$ and $\exists j \in \{1, \ldots, d\},\ p.x_j q.x_j$.*

Definition (Skyline). *The skyline objects of a dataset P in a d-dimensional data space, denoted by SKY(P), is a set of objects that are not dominated by any other objects in P*

$$SKY(P) = \{p \ \in P \mid \nexists\ q \in P, q \prec p\}. \tag{1}$$

The existing distributed and parallel skyline approaches are motivated by the additivity property of the skyline operator. Skyline objects of a dataset can be obtained by a two-step computation: (i) partition the dataset into multiple subsets and find local skyline objects in each subset; (ii) calculate the skyline objects of the union of the local skyline objects. The additivity property is formally defined as follows:

Definition (Additivity of the Skyline). *Given a dataset P and n subsets P_i such that $P = P_1 \cup ... \cup P_n$, then*

$$SKY(P) = SKY(SKY(P_1) \cup ... \cup SKY(P_n)). \tag{2}$$

3.2 Data Partitioning

In parallel skyline solutions based on the additivity property of skyline queries, an intuitive data partition scheme is random partitioning, in which objects in a partition are randomly selected (their attributes are not considered) [11]. Since a dataset can be evenly divided across partitions by a random partitioning method, the number of skyline objects in partitions are expected to be the same under a uniform data distribution. Due to lack of locality information, this approach does not minimize the output of local skyline computation, which would significantly increase the communication cost between local and merging skyline computation components.

A grid partitioning scheme that partitions the data space into multiple grids was employed in parallel and distributed computing environments [34], [35]. By utilizing the domination relation among partitions, objects in a partition that is dominated by another partition can be discarded without skyline calculation. A grid partition can be represented by $V^G = \{R_1^G, \ldots, R_d^G\}$, where R_i^G denotes a value range $[l_i^G, u_i^G]$ in the ith dimension.

To minimize the local skyline results, an angle-based partitioning method maps datasets from a Cartesian coordinate space to a hyperspherical space and groups objects based on their angles to the origin [20], [33]. An object in the hyperspherical space can be represented as $p = \{r, \phi_1, \ldots, \phi_{d-1}\}$. An angle-based partition is denoted by $V^A = \{R, R_1^A, \ldots, R_{d-1}^A\}$, where R is a range $[l_r, u_r]$ of distance to the origin and R_i^A is an angle range $[l_i^A, u_i^A]$ in the ith dimension.

Under angle-based partitioning, the pruning power of an object can be estimated by the volume of the dominance region (area in a two dimensional space). If objects are uniformly distributed in a two-dimensional space, the pruning power of object p in partition V^A is defined as:

$$PP(p, V^A) = 100 * \frac{PArea(p, V^A)}{Area(V^A)}, \tag{3}$$

where $PArea(p, V^A)$ indicates the area of the dominance region of p within the partition, and $Area(V^A)$ denotes the total area covered by the partition. The definitions of $PArea$ and $Area$ can be found in [33].

3.3 MapReduce Overview

The MapReduce framework provides developers with two primitives, map and $reduce$ functions. Under the framework, developers are allowed to focus on their functionality in the customized map and $reduce$ functions. All the issues related to task scheduling, load balancing, and fault tolerance are processed by the framework, which significantly simplifies the parallel program development. By following the primitives, all inputs and outputs are abstracted as key/value pairs. The map function takes key/value pairs from an input stream and produces intermediate results in another key/value pair. The $reduce$ function retrieves the intermediate results from map functions and writes the final results to an output stream. In the shuffle phase, the framework groups and sorts the intermediate results by their keys, and moves them from map functions to $reduce$ functions. The two primitives can be represented as: map(K_1, V_1)→ list(K_2, V_2) and reduce(K_2, list(V_2))→list(K_3, V_3).

4 Two-Phase MapReduce-Based Solution

In this section, we propose our advanced parallel skyline solution using the MapReduce framework.

4.1 Framework Overview

Our solution consists of two MapReduce phases. In particular, we first set up skyline filtering and angle-based partitioning modules in the first phase, which filters out unqualified objects from the input dataset and partitions the processed objects by their angles. In the second phase, map functions calculate local skyline objects on each angle-based

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:33:12 UTC from IEEE Xplore. Restrictions apply.

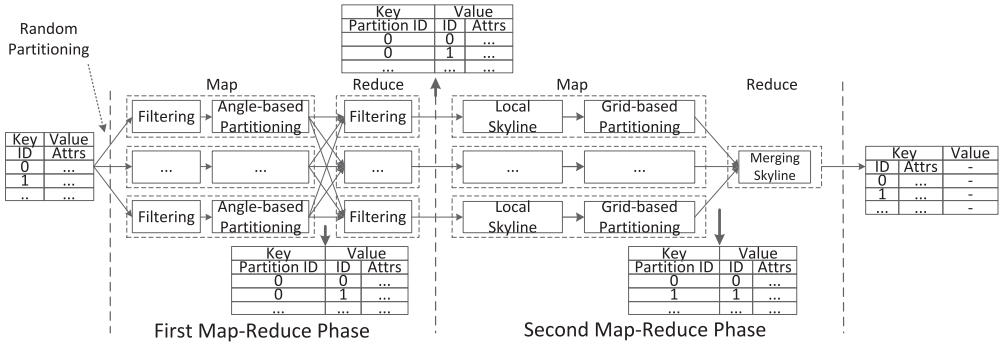


Fig. 2. An overview of the parallel skyline processing using MapReduce.

partition; a reduce function merges intermediate results and produces global skyline objects. Any non-index skyline methods can be easily employed in local and merging skyline computation. Due to high complexity of index creation and difficulty of index sharing among map and reduce functions, index-based techniques are not considered in our solution. The angle-based partitioning method in the first phase is conducive to minimizing the local skyline set.

Fig. 2 illustrates the work flow of our advanced solution, which receives a large dataset as input and produces skyline objects after a two-phase MapReduce computation. Initially, the dataset is evenly split by a random partitioning scheme. Because the skyline filtering and partitioning procedures are I/O bound, and the splits are identical in size, map functions are load-balanced in the first phase. Objects are abstracted as $< id, attrs >$ key/value pairs as input in the map functions, where id is the unique identifier for the object, and $attrs$ are object attributes used in skyline computation. The map functions produce the associated angle-based partition V^A as the key and the object as the value of output.

After the shuffle phase, the objects produced by map functions are grouped and sent to reduce functions for further processing. An object from a map function could be dominated by other objects from another map function. The reduce functions output objects in $< V^A, < id, attrs > >$ key/value pairs. The filtering methods used in the map and reduce functions are not necessarily the same. The number of map and reduce functions are determined by the number of input splits and angle-based partitions $|V^A|$, respectively. The details of the first phase will be described in Section 4.2.

In the second MapReduce phase, a general parallel skyline approach is applied. Each map function calculates local skylines on an angle-based partition and the reduce function merges the results. Any non-index skyline algorithm can be used in the two functions. However, existing approaches cannot be efficiently incorporated into the MapReduce framework. BNL takes an unnecessarily large number of data scans. SFS requires an external sort method to sort objects in advance, which suffers from expensive I/O accesses. Therefore, the map functions partition objects by a grid-based partitioning method after local skyline calculation. The input and output of map functions are $< V^A, p >$ and $< V^G, p >$, where V^A and V^G are the associated angle-based and grid-based partitions of the object p. With the grid partition information, the reduce function merges skyline objects by applying our Partial-presort Grid-based Partition Skyline algorithm. The number of map functions is determined by the number of angle-based partitions $|V^A|$. The details of the second phase will be described in Section 4.3.

Fig. 2 displays that the random, angle-based, and grid-based partitioning are applied in our approach. It is worth mentioning that there is no need for the random partitioning scheme to scan datasets. An input data is partitioned into equal sizes by a straightforward calculation (i.e., calculating offsets of data partitions in data streams). Moreover, the additional MapReduce phase and data partitioning are proposed for optimization at local and merging skyline computation, which are the major bottlenecks of parallel skyline evaluation.

4.2 Filtering Phase

In the first phase, the map functions filter out unqualified objects and partition the processed objects by their angles to the origin. The reduce functions further reduce the size of the dataset. Fig. 3 shows a concrete example in the first MapReduce phase. The data space is partitioned into three angle-based sub-spaces. Initially, there are ten objects in the input dataset. The dataset is evenly split and processed by three map functions in parallel. In the top map function, objects $\{a, b, c, d\}$ are sequentially processed. After the skyline filtering, objects $\{b, d\}$ are filtered out because they are dominated by objects $\{a, c\}$. Then, the first map function calculates the angle-based partitions of objects a and c and outputs $< 2, a >$ and $< 0, c >$. Then, three reduce functions are executed in parallel, each of which processes a partition. The top reduce function takes object c from the first map function and object g from the second map function. After the filtering check, object c is discarded. The details of the angle-based partitioning method can be found in [33].

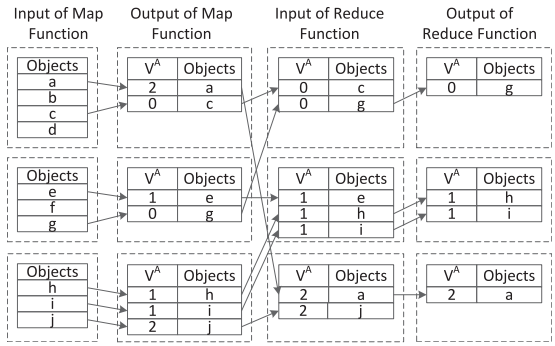


Fig. 3. An example of the first MapReduce phase.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:33:12 UTC from IEEE Xplore. Restrictions apply.

4.2.1 Progressive Filtering Method

Filtering methods that set up a set of filtering objects have been studied in [19], [12]. When retrieving a new object, the methods compare the object with the filtering objects. The object will be produced as a skyline candidate if it is incomparable with the filtering objects; otherwise, the object will be eliminated. The filtering objects will be replaced if a better filtering object is found.

In the existing filtering methods, such as MaxSum and MaxDist, filtering objects can be selected in advance and dynamically updated during processing. However, due to inaccessibility of the input dataset, the filtering objects cannot be pre-determined in our MapReduce-based solution. Scanning the entire dataset is considerably expensive. Therefore, we design a progressive filtering method, in which the filtering object set is initially empty and progressively filled with objects that have been scanned. Moreover, the filtering objects can be dynamically replaced by newly found objects under a particular replacement strategy. The progressive filtering method may produce more objects than the ones that initially set up filtering objects, because the best filtering objects cannot be found until the end of the input stream is reached; however, the progressive method can be completed in a single data scan. The progressive filtering method described in Algorithm 1 can be used in both map and reduce functions. The function *Progressive Filtering* receives an object and returns whether the object is dominated by the filtering objects.

Algorithm 1. Progressive Filtering

1: $FilteringObjects \leftarrow \emptyset$
2: **function** ProgressiveFiltering$p$
3: **if** $\exists\ q \in FilteringObjects,\ q \prec p$ **then**
4: Return True
5: **else**
6: Update $FilteringObjects$
7: Return False
8: **end if**
9: **end function**

4.2.2 Partition-Aware Replacement Strategies

In line 6 of Algorithm 1, the filtering objects are updated by a particular replacement strategy. Theoretically, maximizing the fused dominance region can obtain the best data reduction rate (DRR) in a dataset with uniform distribution [12]. Two heuristic methods, MaxSum and MaxDist, were proposed to avoid the high cost of calculating the volume of the fused dominance region. MaxSum maximizes the sum of the volume of dominance regions, while MaxDist maximizes the distances among filtering objects.

We set up a set of preliminary experiments to analyze object distribution after the filtering process. We apply MaxSum and MaxDist on uniform, cor-related, and anti-related datasets. Each dataset contains four million objects in a two dimensional space. The number of filtering objects is set to 16. The data space is evenly partitioned into four angle-based partitions. The experimental results are displayed in Table 2.

We observe that both methods perform well on the three datasets in terms of data reduction rate. For the uniform and cor-related datasets, only tens or hundreds of objects are produced in a partition, on which the skyline computation can be completed in a short time. On the other hand, for the anti-related dataset, MaxSum and MaxDist output many more objects in partitions 1 and 4 than partitions 2 and 3. If each partition will be processed by a map function in the second MapReduce phase in parallel, processing the *fattest partition* that has the greatest number of objects would be the bottleneck, because the execution time of a parallel program is determined by the slowest process, and skyline algorithms take a longer time on larger datasets.

TABLE 2
Preliminary Results of MaxSum and MaxDist

Methods	DBs	Partitions				Total
		1	2	3	4	
MaxSum	Uni	310	3	7	169	489
	Cor	9	3	3	6	21
	Anti	835 K	658 K	651 K	836 K	3.0 M
MaxDist	Uni	825	7	3	424	1,259
	Cor	6	3	3	6	18
	Anti	800 K	670 K	640 K	802 K	2.9 M

Thus, we propose two novel partition-aware filtering methods that are able to not only reduce the number of objects in the fattest partition, thereby shortening the execution time of parallel local skyline computation in the second MapReduce phase, but also obtain better or comparable data reduction rate than MaxSum and MaxDist. Selecting filtering objects among partitions will increase the volume of their fused dominance region. In particular, our methods dynamically replace the filtering objects based on the population of objects among partitions.

Algorithm 2. Aggressive Partition-Aware Filtering

1: **function** APF(K, $PList$)
2: $FilteringObjects \leftarrow \emptyset$
3: $N = PList$.Length
4: Sort $PList$ based on population in descending order.
5: m = min(K, N)
6: **for** i = 0 to m-1 **do**
7: Select the best object from $PList[i]$.
8: Append the object to $FilteringObjects$.
9: **end for**
10: **if** $K \geq N$ **then**
11: Select the best $(K - N)$ objects from the fattest partition, besides the one selected in line 7.
12: Append these objects to $FilteringObjects$.
13: **end if**
14: Return $FilteringObjects$
15: **end function**

Aggressive partition-aware filtering (APF). APF aggressively eliminates objects in the fattest partition. This method selects filtering objects by considering the object distribution among partitions. Particularly, APF selects one filtering object from each partition, from the fattest to the thinnest. If the number of filtering objects K is greater than the number of partitions N, the additional $(K - N)$ filtering objects are selected from the fattest partition for slowing down its population growth. The details are described in Algorithm 2.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:33:12 UTC from IEEE Xplore. Restrictions apply.

APF takes the number of filtering objects K and a list of partition information $PList$, and returns a filtering object set. Each record in $PList$ keeps popularity information as well as the best K objects found in a partition. Initially, the population is set to be zero and incremented by one when a new qualified candidate is detected in the partition. The best K objects are updated during the filtering process. The time complexity of APF is $O(N\log N + K)$.

Due to the unpredictability of objects that will be retrieved subsequently, efforts of eliminating objects in the current fattest partition would be inefficient if the partition will not be the fattest one at the end. During the filtering process, considering current object distribution and predicting subsequent object distribution at the same time is another promising method. Thus, we propose an alternative filtering method that proportionally eliminates objects.

Proportional partition-aware filtering. PPF proportionally selects filtering objects from partitions based on object distribution. The more processed objects remain in a partition, the more attention of object elimination will be paid in the partition. By dynamically varying the filtering point distribution, PPF is able to proportionally slow down the population growth of all partitions with an assumption that any partition would possibly be the fattest one at the end. The details of PPF are described in Algorithm 3. The complexity of PPF is $O(N)$.

APF and PPF can be used in mappers and reducers independently. Each mapper and reducer maintains a single copy of $PList$ managing popularity information. $PList$ is not shared among mappers or reducers.

Algorithm 3. Proportional Partition-Aware Filtering

1: **function** PPFK, $PList$
2: $FilteringObjects \leftarrow \emptyset$
3: $N = PList$.Length
4: Let $total$ be the total number of objects in $PList$.
5: $remains = K$
6: **for** $P_i \in PList$ **do**
7: Let $obnums$ be the number of objects in P_i
8: $m = \lfloor \frac{obnums}{total} \times K \rfloor$
9: Select the m best objects from P_i.
10: Append these objects to $FilteringObjects$.
11: $remains = remains$ - m
12: **end for**
13: Let P_j be the fattest partition.
14: Select the best $remains$ objects from P_j, besides the ones selected in line 9.
15: Append these objects to $FilteringObjects$.
16: Return $FilteringObjects$
17: **end function**

4.2.3 The Best Filtering Object Selection in a Partition

Theoretically, the object with the maximum pruning power in the partition will be selected as the best filtering object (see lines 7 and 11 of Algorithm 2, and lines 9 and 14 of Algorithm 3). In a uniform dataset, the pruning power of an object can be approximated by the volume of its dominance region in the partition. Calculating the pruning power in a d-dimensional space is expensive. Equation (3) shows the formula of the pruning power in a two-dimensional space.

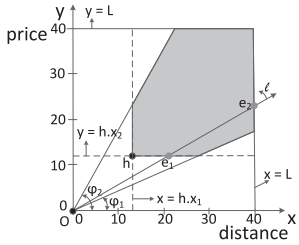


Fig. 4. Pruning power in an angle-based partition.

A method that calculates the volume of partitions in a d-dimensional space was reported in [33]; however, calculating the volume of the dominance region in a particular partition is non-trivial.

In the two dimensional space $[0, L]^2$, as shown in Fig. 4, the shaded area is the dominance region of object h in the central partition. We create a ray l from the origin that intersects with the dominance region $[h.x_i, L]^2$ at points e_1 and e_2. The dominance region can be fully covered by the area that we sweep the ray from ϕ_1 to ϕ_2. Thus, the area of the dominance region can be represented by $\int_{\phi_1}^{\phi_2} l_{e_1e_2}(h, \phi)d\phi$, where $l_{e_1e_2}(h, \phi)$ is the length of the line segment from e_1 to e_2. In fact, the ray l may intersect with the boundary of the dominance region $[h.x_i, L]^2$ at four points, which are the intersections of l and lines $x = h.x_1$, $y = h.x_2$, $x = L$, and $y = L$. We call the first two points *lower intersections* (l intersects with the lower bounds of the dominance region in the two dimensions), and the latter two points *upper intersections*. e_1 is the lower intersection with greater coordinate values. e_2 is the upper intersection with smaller coordinate values. To extend it to a d-dimensional space, the Volume of Dominance Region ($\widetilde{VDR}$) of an object h in an angle-based partition P is

$$\widetilde{VDR}(h, P) = \int_{P.R_1^A.l}^{P.R_1^A.u} \ldots \int_{P.R_{d-1}^A.l}^{P.R_{d-1}^A.u} l_{e_1e_2}(h, \phi_1, \ldots, \phi_{d-1}) d_{\phi_1} \ldots d_{\phi_{d-1},} \quad (4)$$

where $l_{e_1e_2}(h, \phi_1, \ldots, \phi_{d-1})$ denotes the length of the intersection between a ray l starting from the origin to the direction $\{\phi_1, \ldots, \phi_{d-1}\}$ and a hypercube $[h.x_i, L]^d$. e_1 is the lower intersection with the greatest coordinate values, and e_2 is the upper intersection with the smallest values. $[P.R_i^A.u, P.R_i^A.l]$ indicates the range of angles of the partition P in the ith dimension.

Basically, the computation complexity of Equation (4) based on interpolation methods is $O(K^d)$, where K is the complexity of one-dimensional integration computation and proportional to the number of sample points selected by the interpolation methods. Moreover, the computation complexity grows exponentially when the number of dimensions increases. The cost of calculating the fused dominance region in a particular partition would be even higher (See Fig. 5).

A heuristic partition-aware filtering object selection method. To avoid the high cost of the fused dominance calculation, we turn to a heuristic alternative, in which the partition of the object and its dominance region in the entire space are considered. We assume that an object would have greater pruning power in its partition than the ones from other partitions. If two objects are in the same partition, the one with

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:33:12 UTC from IEEE Xplore. Restrictions apply.

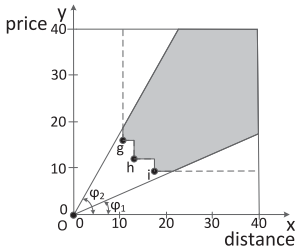


Fig. 5. Fused dominance region in an angle-based partition.

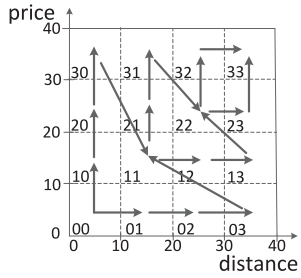


Fig. 6. Grid-based partition partial order in a two dimensional space.

greater pruning power would be better than the other. The pruning power is evaluated by using their fused dominance region in the entire space rather than in a particular partition. The difference from MaxSum is that an object's partition is taken into account with higher priority than the dominance region in filtering point selection. Our heuristic method can be formally described as follows. $\widetilde{VDR}(p_i)$ denotes the VDR of p_i in the entire space [12].

Rule. Given two points p_i, p_j in a partition P, p_i will be first selected as a filtering point if

1) $p_i \in P, \quad p_j \notin P$ **OR**
2) $p_i, p_j \in P, \quad \widetilde{VDR}(p_i) > \widetilde{VDR}(p_j)$

It is worth noting that the heuristic method compromises data reduction rate in the following cases: 1) all filtering objects are selected from partition P_i to eliminate objects in P_i; however, it is possible that a better filtering object could be found in another partition P_j, because the object has a larger dominance region in P_i; 2) a larger dominance region in the entire space does not necessarily indicate a larger dominance region in a certain partition; 3) the filtering objects with the greatest sum of volume of dominance regions may not be the ones with the greatest volume of the fused dominance region.

4.3 Skyline Processing Phase

In the second MapReduce phase, the skyline computation is processed by an efficient method. Each map function works on objects in an angle-based partition. The reduce function merges intermediate results and produces skyline objects. Due to lack of indices, an index-independent method, such as BNL [7] and SFS [10], can be applied in local and merging skyline computation.

In Fig. 2, we observe that the merging skyline computation is processed in a single reduce function. Neither setting up more computing resources nor increasing the number of data partitions would help to improve the merging process. Thus, the efficiency of merging local skyline objects is critical to the performance of our proposed solution.

As the size of dataset increases, the skyline candidates received by the reduce function cannot be entirely loaded into memory. Instead, parts of objects have to be temporarily stored in files. BNL keeps skyline candidates in a window. If the window is full, other candidates are written to a temporary file. After an entire scan, the objects in the window are output as skyline objects and the ones in the file are loaded and processed in the next iteration. The process continues until no temporary file is generated. The number of iterations relies on the window size and the order of objects retrieved from the input stream. SFS, LESS [17], and SaLSa [5] work in a sort-based manner. They require the input objects to be pre-sorted in advance. Sorting a large dataset by using an external sort algorithm suffers from many I/Os. Another skyline method that utilizes D&C on external memory also assumes that input objects have been sorted in order [31].

To reduce the number of passes in BNL and avoid the expensive external sorting in sort-based methods, we propose a novel Partial-presort Grid-based Partition Skyline algorithm that is able to reduce object comparisons and I/O costs in merging skyline computation. In our method, objects are first re-arranged in a particular order so that the objects in a partition cannot be dominated by any object retrieved from subsequent partitions. Then, the following benefits can be obtained in skyline calculation.

1) The objects in a partition close to the origin will be scanned first, which reduces the number of object comparisons, object replacements, and iterations.
2) If a partition is dominated by an object, all objects in the partition can be eliminated without comparison. If the input stream can be randomly accessed, such as a file stream, all objects in the partition are skipped by jumping to the first object in the next partition, which can save time on data retrieval.
3) With the grid partition information, the size of temporary files can be reduced by selecting sentinel objects that are used for filtering out objects in the currently scanned partition.

4.3.1 Partial-Presort Grid-Based Partition Skyline

Before illustrating our algorithm, we introduce the definitions used in our method. As mentioned before, a grid partition V^G can be denoted by $\{R_1^G, \ldots, R_d^G\}$, where R_i^G is a range in the ith dimension. The identifier of V^G can be represented by $\{r_1^G, \ldots, r_d^G\}$, where r_i^G is a partition index in the ith dimension. For example, the identifier of $V^G = \{[0, 10], [0, 10]\}$ is 00, which is denoted as P_{00} (see Fig. 6).

Definition (Partition domination). *Given two partitions P_i and P_j, $P_i \prec P_j$ iff $\forall p_m \in P_i, p_n \in P_j, p_m \prec p_n$.*

Corollary 4.1. *Given two partitions P_i and P_j,*

$$P_i \prec P_j \iff P_i.r_k^G < P_j.r_k^G, \ 0 \le k \le d. \tag{5}$$

Take Fig. 6 as an example, P_{00} dominates P_{11}; thus, the partition index zero (0) of P_{00} is smaller than the one (1) of P_{11} in both dimensions.

Property (Closeness of partition). *The closeness of a partition to the origin describes the likelihood of objects in the partition*

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:33:12 UTC from IEEE Xplore. Restrictions apply.

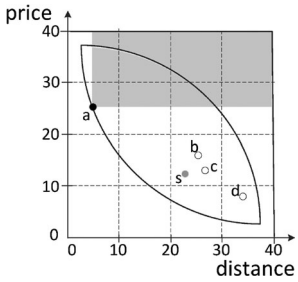


Fig. 7. Sentinel objects.

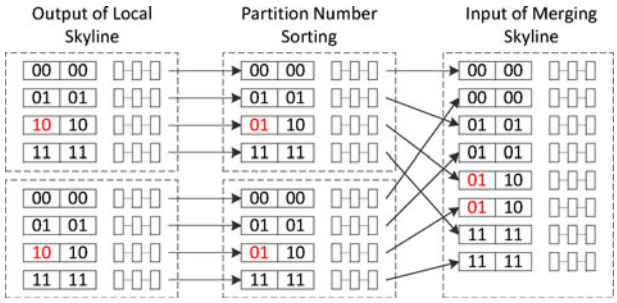


Fig. 8. Build the partition partial order in the shuffle phase.

to be skyline objects. The closeness is evaluated by a combination of partition indices in the data space. In a particular dimension, the smaller the index is, the lower the possibility of objects in the partition being dominated by objects in other partitions will be. Given two partitions P_i and P_j, we compare the closeness of the partitions in two steps: 1) sort the index value of a partition in ascending order; 2) compare the index values one by one. The first index is the most significant digit after sorting. An example will be illustrated in Fig. 8 later.

Property (Grid-based Partition partial order). *Partitions are sorted by their closeness to the origin in ascending order. All objects in the closest partition are retrieved first. The order of objects in a particular partition is not enforced. An example of grid-based partial order in two dimensions is displayed in Fig. 6. Partitions are scanned in a multi-way Z-order manner. P_{00} is the first partition, and then P_{01} and P_{10}. The implementation will be illustrated in the next section.*

Our Partial-presort Grid-based Partition Skyline algorithm takes a set of objects that have been sorted in grid-based partition partial order, and produces skyline objects. The approach caches skyline objects in a window. We maintain a tuple $< hasDominated, hasVisited >$ with initial value $< 0, 0 >$ for each partition, where $hasDominated$ indicates whether the partition has been dominated by an object found so far. If the partition has been dominated, all objects in the partition can be discarded immediately. $hasVisited$ records whether there is an object that has been scanned in the partition. When reaching the first object in a partition, the corresponding $hasVisited$ is set to 1. Then, we update $hasDominated$ of all other partitions. If a new object is incomparable with objects in the window, the object is appended to the window if it is not full; otherwise, the object is written to a temporary file. All objects in the file will be processed in the next iteration. The process is continued until no temporary file is generated.

Moreover, the efficiency of our method relies on the number of objects eliminated by objects in the window. The methods that dynamically replace objects in the window have been proposed as an improvement to the basic BNL. However, appropriately selecting the objects is difficult due to the unpredictability of subsequent objects. For example, if objects {a, b, c, d, s} in Fig. 7 are retrieved and the window size is 2, keeping objects a and d in the window does not help eliminate objects b and c. Thus, we make use of sentinel objects for the purpose of object elimination check in our method. Because partitions are sequentially scanned, we maintain sentinel objects for each partition. The sentinel objects are reset when a new partition is reached. All objects are compared with objects in the window and sentinel objects. The sentinel objects will be replaced if any of them is dominated by a newly found object. Sentinel objects along with the ones in the window are employed to filter out unqualified skyline candidates; nevertheless, their difference is that the sentinel objects are selected from the partition being scanned, whereas the objects in the window may distribute across the global space. The sentinel objects are not written to the temporary file after an iteration. In the previous example, if s is set to be a sentinel object, objects b and c can be eliminated. The details of our method are illustrated in Algorithm 4. The complexity of $PGPS$ is $O(|P|^2)$, where $|P|$ denotes the size of the input dataset.

Algorithm 4. Partial-Presort Grid-Based Partition Skyline

```
1: function PGPS(Input, Output)
2:   while True do
3:     window ← ∅; sentinel ← ∅
4:     Initialize hasVisit and hasDominated for partitions.
5:     InputFile = Input
6:     Let Input be a temporary file.
7:     Return if InputFile does not exist.
8:     for Each object o in InputFile do
9:       Let V^G be the partition identifier of o.
10:      if hasDominated[V^G] = 1 then
11:        Jump to the next partition
12:        Continue
13:    end if
14:    if hasVisit[V^G] = 0 then
15:      hasVisit[V^G] = 1
16:      Update hasDominated of other partitions.
17:      Clear sentinel.
18:    end if
19:    if ∃ o' ∈ window ⋃ sentinel, o' ≺ o then
20:      Continue
21:    end if
22:    if ∃ o' ∈ window ⋃ sentinel, o ≺ o' then
23:      Eliminate o'.
24:    end if
25:    Insert o into sentinel if it is not full.
26:    if window is not full then
27:      Insert o into window.
28:    else
29:      o is written to Input.
30:    end if
31:  end for
32:  Write objects in window to Output.
33: end while
34: end function
```

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:33:12 UTC from IEEE Xplore. Restrictions apply.

4.3.2 Presort Processing

Throughout the following four steps, the presorting process is accomplished by an effective approach in two data scans. First, we create a temporary file for each grid partition and write objects to the files based on their partitions. The sizes of the files are recorded, so that they can be used as offsets when PGPS jumps over a partition in the input data stream (see line 11 of Algorithm 4). Then, all temporary files are sorted by the grid-based partition partial order. Finally, objects are read from files in order.

The presorting can be easily implemented in the following four steps in the shuffle phase of the MapReduce framework. First, each map function groups local skyline candidates by their grid-based partitions after local skyline calculation. Take Fig. 7 for example, object a is a skyline candidate in partition P_{20} and its partition index is <20>. Second, all partition indices are duplicated. The first copy of indices is sorted in an ascending order (e.g., <1,0> is changed to <0,1>). Third, all partitions are sorted by their two copies of partition indices. The first copy is more significant than the second copy. Last, the reduce function reads objects from a top partition to a bottom one. Fig. 8 displays an example of the presort process in a two dimensional space. The first copy of indices represents the order of the closeness of partitions (P_{01} and P_{10} are identically close to the origin); the second copy of indices preserves the partition information. Without the second copy of indices, objects in P_{01} and P_{10} would have been mixed in the input stream to the reduce function.

4.4 PGPS, BNL, and SFS

PGPS works similarly to BNL and SFS. BNL keeps a window and compares objects in the window with every other object. Object replacement is considered because objects in the window may be dominated by new objects. No preliminary sort is required by BNL; whereas PGPS sorts objects by their grid-based partitions in advance. With the grid-based partition partial order, the advantage of PGPS is that the number of passes is reduced by first receiving better objects from the input stream. Objects cannot be dominated by the ones in subsequent partitions. In addition, objects in the partition that have been dominated by preceding objects can be discarded. Skipping these objects by jumping over the input stream can save time on data retrieval. Furthermore, setting up sentinel objects can not only eliminate more objects in iterations but also reduce the I/O cost on writing objects to temporary files and reading them back in the next iteration.

On the other hand, SFS assumes that input objects have been sorted in order according to their values in a particular dimension. Due to the presort processing, objects cannot be dominated by any subsequent objects. Object replacement is not required in SFS. PGPS is different from SFS in that PGPS sorts objects in a coarse-grained grid-based partition partial order, which suffers from much less I/O overhead on presorting. In addition, the order in PGPS is created by considering the object values in all dimensions rather than only one dimension. With grid partition information, optimizations can be applied in PGPS.

More importantly, PGPS is the most appropriate algorithm that can be incorporated in the MapReduce framework. The shuffle processing works similarly to the presorting in PGPS. The only extra overhead induced by PGPS is that the keys (the partition identifiers) have to be sorted in order, which can be completed by an in-memory sort approach. When reaching the reduce function, Algorithm 4 can be used to merge local skyline results.

5 Experimental Validation

In this section, we first evaluate our proposed filtering methods and PGPS on a single node. Then, we evaluate the two advanced parallel skyline solutions on Hadoop 1.0.3, which is an open source implementation of the MapReduce framework [1]. APF-PGPS and PPF-PGPS use APF and PPF as filtering methods in the proposed framework, respectively. PGPS is applied in both solutions as the merging skyline method. Moreover, we also implemented the MR-GPMRS solution [26] and a single MapReduce-phase solution, S-MS-BNL [8], [36], as two baseline solutions. MaxSum and BNL algorithms are used as the filtering and merging skyline methods in S-MS-BNL. BNL is also applied in local skyline calculation in the four solutions. All solutions were implemented in Java.

We extended the map and reduce functions with an additional boolean parameter, which is used to indicate whether the input stream exhausts. The two primitives can be represented as: map(K_1, V_1, $isLast$)→ list(K_2, V_2) and reduce(K_2, list(V_2), $isLast$)→list(K_3, V_3). If the parameter $isLast$ is set, the last object is received in the first iteration. Before exiting the functions, more iterations could be conducted if necessary.

The experiments were conducted on a twelve-node shared-nothing cluster. Each node is equipped with one Intel Xeon 2.4 GHz processor, 2 GBytes of memory. All nodes are connected by GigaBit Ethernet network. We utilized the default configuration in Hadoop. The window size was set to be 1,000 objects. We randomly generate uniform and anti-related datasets in a five dimensional space. The parallel solutions are executed with eight nodes by default.

5.1 Efficiency of Partition-Aware Filtering Methods

We evaluate the execution time, effectiveness of filtering, and data distribution of MaxSum, MaxDist and our partition-aware filtering methods on anti-related datasets. The results on the uniform and correlated datasets are not shown due to the similar performance of these filtering methods and the space limit in this paper. We randomly generate anti-related datasets that have 0.8 and 1.6 million objects in a five dimensional space. The size of the datasets are 32 and 64 MBytes, which are the sizes of basic units commonly processed by mappers in MapReduce systems. The number of filtering objects is set to 16. In our filtering methods, the data space is partitioned into eight angle-based sub-spaces.

Table 3 shows the comparison of APF and PPF with MaxSum and MaxDist. All methods set up the filtering object set progressively during data scans. The average number of objects in partitions, as well as the total number of objects, produced by the methods are listed. APF and PPF are able to eliminate more objects because filtering objects are selected from all partitions. The fused dominance region of

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:33:12 UTC from IEEE Xplore. Restrictions apply.

TABLE 3
Filtering Method Comparisons on Anti-Related Datasets

Data	Methods	Partitions (K)								AVG (K)	MAX (K)	Total (K)	SD (K)	ET (sec)
		1	2	3	4	5	6	7	8					
32 MB	MaxSum	67	34	25	23	23	23	26	55	35	67	276	45	3.25
	MaxDist	85	53	64	65	64	64	53	89	67	89	537	35	9.73
	APF	47	29	25	24	24	24	29	47	31	47	249	27	3.45
	PPF	42	27	23	22	23	24	28	42	29	42	231	23	3.90
64 MB	MaxSum	117	55	46	45	45	48	63	141	70	141	560	99	6.25
	MaxDist	142	87	93	92	89	91	105	133	104	142	832	57	16.36
	APF	95	62	49	46	45	47	56	95	62	95	496	56	6.18
	PPF	89	64	57	55	54	56	62	89	66	89	525	38	6.49

the filtering objects will be larger than that of the filtering objects clustered in a central area based on the individual dominance area in MaxSum. The "Max" column shows the number of objects existing in the fattest partition. If each partition is processed in parallel, the execution time will be determined by the processing on the largest dataset. Due to awareness of population in partitions, APF and PPF can slow down the growth of the "Max" value by setting up more filtering points in the fattest partition and potentially reduce the execution time of parallel local skyline calculation in the second MapReduce phase. The "ET" column displays the execution time of the filtering process. APF and PPF are comparable to MaxSum in execution time because the filtering methods are I/O-bound. The execution times of MaxSum, APF, and PPF largely depend on I/O throughput of disks rather than the efficiency of filtering algorithms. The computational complexity of MaxDist is much higher than the other three methods [12].

5.2 PGPS Skyline Algorithm Evaluation

We compare PGPS with BNL and SFS by varying the size of datasets. We randomly create uniform and anti-related datasets that have 1.6 (64 MBytes), 3.2 (128 MBytes), 6.4 (256 MBytes), and 12.8 (512 MBytes) million objects in a five dimensional space. A distributed sorting program in Hadoop is adopted in the presorting process in SFS.

As the "presorting" column shows in Table 4, BNL does not require presorting. PGPS can complete presorting in around 4 seconds on the 64 MBytes uniform dataset and 51 seconds on the 512 MBytes uniform dataset, which are 10 to 92 percent of the time needed by SFS. In the skyline calculation, the performance of the three methods is close on uniform datasets, whereas SFS runs nearly 2.4 and 1.4 times faster than BNL and PGPS on anti-related datasets. Before skyline calculation, SFS assumes that the input objects have been sorted by their values in a particular dimension, whereas PGPS presorts objects by their partitions. In the grid-based partition partial order, a given object cannot be dominated by objects in subsequent partitions but would be dominated by an object in the same partition. Thus, PGPS, in terms of the execution time of skyline calculation, as well as the number of iterations and size of temporary files, always stands between SFS and BNL.

More importantly, PGPS is the fastest merging skyline algorithm in our proposed framework. The column "Merging Skyline in MapReduce" indicates the execution time if the algorithm is implemented as the merging skyline method in our framework. BNL requires the same time indicated in the "Skyline" column to merge local skyline candidates. The presorting of SFS is processed as a part of merging skyline calculation. The execution time of merging skyline calculation by SFS is the total time of its presorting and skyline calculation. Since the shuffle phase is an I/O-bound process, in which data will be retrieved from disks and transferred on network, the overhead of the two in-memory sorts in PGPS presorting is lower than that of the I/O or network communication by one or two orders of magnitude. The execution time of merging skyline calculation by PGPS can be estimated as the time of skyline calculation. Based on the above measurement, PGPS performs around 43 percent faster than BNL and 3.6 times faster than SFS on the uniform and anti-related datasets.

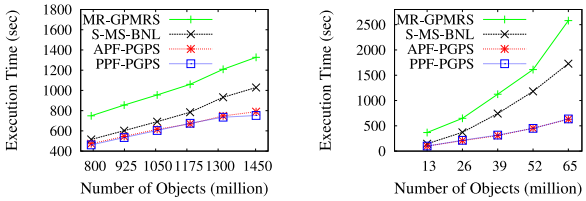
5.3 Scalability with Cardinality

In the experiments, we evaluate the effect of data cardinality on our advanced MapReduce-based solutions in a five dimensional space. The cardinality of uniform datasets varies from 792 to 1,452 million objects, whereas that of

TABLE 4
Comparison of Skyline Algorithms

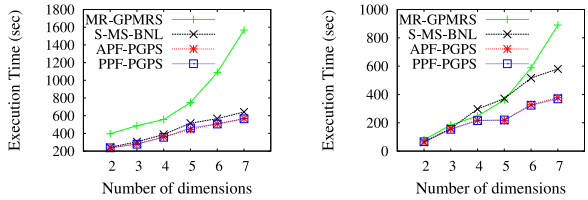
Data	Methods	Presort/Skyline/Merging Skyline in MapReduce (seconds)				Number of iterations				Size of temporary files (M Bytes)			
		64	128	256	512	64	128	256	512	64	128	256	512
Uni	BNL	0/6/6	0/11/11	0/20/20	0/37/37	70	82	104	126	11	15	26	37
	SFS	40/4/44	44/8/52	49/15/64	55/28/83	24	26	34	38	1.9	2.5	4.1	5.0
	PGPS	4/5/5	11/9/9	23/17/17	51/34/34	26	30	41	45	2.7	3.6	6.5	8.7
Anti	BNL	0/23/23	0/58/58	0/146/146	0/375/375	194	291	442	674	123	317	880	2,457
	SFS	42/8/50	44/23/67	52/69/121	61/177/238	112	168	266	415	67	185	579	1,740
	PGPS	4/14/14	8/34/34	19/88/88	48/206/206	125	191	302	474	88	248	743	2,252

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:33:12 UTC from IEEE Xplore. Restrictions apply.



(a) Uniform datasets. (b) Anti-related datasets.

Fig. 9. Varying dataset cardinality.



(a) Uniform datasets. (b) Anti-related datasets.

Fig. 10. Varying dataset dimensionality.

anti-related datasets is up to 65 million objects. As Fig. 9 displays, the execution time of all solutions increases when the datasets grow. MR-GPMRS is the slowest one, and the reason is three-fold: (1) APF, PPF, and MaxSum filter out more objects than a coarse-grained partition-based filtering method; MR-GPMRS receives larger input datasets for local skyline calculation than the other three solutions; (2) MR-GPMRS has large input datasets in merging skyline calculation; and (3) the independent partition group generation, independent group merging, and duplicate elimination cannot be processed in parallel, which become expensive to skyline processing in a high-dimensional space. Table 5 shows the sizes of inputs of local/merging skyline calculation in APF-PGPS, PPF-PGPS and MR-GPMRS. Take the uniform dataset of 800 million objects for example, MR-GPMRS processes the dataset in original size; APF and PPF filtering methods reduce the datasets to 4.4 and 3.9 million objects before local skyline processing in APF-PGPS and PPF-PGPS. At the merging stage, MR-GPMRS has to merge 251 thousand local skyline candidates from 470 mappers; while APF-PGPS and PPF-PGPS only generate 2.6-2.7 thousand local candidates from 8 mappers. Moreover, APF-PGPS and PPF-PGPS execute 9 to 27 percent faster than S-MS-BNL on uniform data-sets, and 45 to 174 percent faster on anti-related datasets. The reason is that although two-phase MapReduce solutions suffer from extra overhead of an additional processing phase, skyline calculation can be completed in a shorter time period, because filtering methods can significantly reduce the number of objects fed into local skyline calculation. These objects are distributed in a balanced manner, so that computing resources can be fully utilized during parallel processing. PGPS is more efficient than BNL in merging skyline calculation.

5.4 Scalability with Dimensionality

We evaluate the performance of the four solutions by varying dimensionality of datasets. The numbers of objects in uniform and anti-related datasets are fixed at 800 and 13 million. In Fig. 10, we observe that, as the dimensionality increases, all solutions take longer time in skyline calculation due to higher costs of object comparison and I/Os (writing objects to temporary files by skyline algorithms). The execution time of MR-GPMRS rapidly increases, because (1) it receives more objects than other solutions in local and global skyline calculation, and (2) the cost of bitstring and independent group maintenance exponentially grows with the number of dimensions. Table 6 compares the sizes of inputs of local/merging skyline calculation in APF-PGPS, PPF-PGPS, and MR-GPMRS in a three or seven dimensional space. The comparison results in a five dimensional space can be found in Table 5. From the results, the proposed filtering methods can largely reduce the sizes of input datasets for local/merging skyline processing. APF-PGPS and PPF-PGPS can discard most of object candidates at the filtering stage, and take only 2.5 and 2.1 million objects in local skyline calculation in a three dimensional space. Moreover, APF-PGPS and PPF-PGPS execute 5 to 14 percent faster than S-MS-BNL on uniform datasets, and 7 to 57 percent faster on anti-related datasets. The growth rate of execution time of S-MS-BNL is higher than APF-PGPS and PPF-PGPS on both datasets, because PGPS can save more in object comparison and I/O access when processing higher dimensional datasets.

5.5 Effect of Number of Nodes

We evaluate the speedup of proposed solutions by scaling up the size of our cluster. The uniform and anti-related datasets are fixed at 800 and 13 million objects in a five dimensional space. In Fig. 11, the execution time of all solutions drop as the size of the cluster increases. The dropping rate on uniform datasets is higher than anti-related datasets. The reason is that the majority of objects in uniform datasets can be eliminated by filtering and calculating local skylines. The execution time of merging skyline calculation is short due to small inputs. Because the filtering and local skyline calculation can be executed in parallel, the solutions are able to run much faster by introducing new nodes. However, a relatively large number of skyline candidates in anti-related datasets are produced by local skyline calculation. The merging skyline calculation contributes to a large

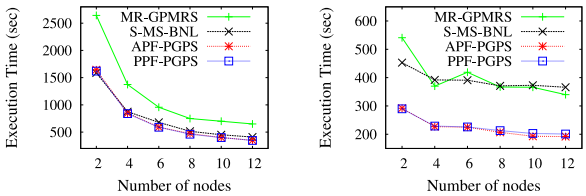
TABLE 5
Size of Inputs in Local and Merging Skyline Calculation by Varying Dataset Cardinality

Methods	Uniform		Anti-related	
	800 M	1,450 M	13 M	65 M
APF-PGPS	4.4M/2.6K	6.7M/5.4K	3.9M/668K	15.7M/2.9M
PPF-PGPS	3.9M/2.7K	6.2M/5.7K	4.0M/731K	16.2M/3.5M
MR-GPMRS	800M/251K	1,450M/448K	13M/903K	65M/6.7M

TABLE 6
Size of Inputs in Local and Merging Skyline Calculation by Varying Dataset Dimensionality

Methods	Uniform		Anti-related	
	3	7	3	7
APF-PGPS	2.5M/2.0K	5.0M/3.7K	1.5M/573K	5.3M/2.7M
PPF-PGPS	2.1M/1.9K	4.5M/3.3K	1.4M/611K	4.9M/2.7M
MR-GPMRS	800M/176K	800M/281K	13M/754K	13M/3.9M

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:33:12 UTC from IEEE Xplore. Restrictions apply.



(a) Uniform datasets. (b) Anti-related datasets.

Fig. 11. Effect of number of nodes.

portion of execution time of the entire processing. Therefore, the performance of the solutions cannot be significantly improved by utilizing more computing resources. When it comes to the anti-related datasets, APF-PGPS and PPF-PGPS perform approximately 55 to 90 percent better than MR-GPMRS and S-MS-BNL in terms of execution time because of more balanced input datasets and improved performance of PGPS.

6 Conclusion

We propose an advanced parallel skyline scheme utilizing the MapReduce framework in this research. In our solution, we design two heuristic partition-aware filtering approaches, APF and PPF, to filter out objects in a distributed manner. By utilizing the shuffle processing in the MapReduce framework, we devise PGPS algorithm that can efficiently calculate skyline objects on large datasets. Moreover, PGPS can be easily incorporated into the MapReduce framework. We demonstrate the efficiency and effectiveness of the proposed solution through extensive experiments.

References

[1] Apache Hadoop [Online]. Available: http://hadoop.apache.org, 2013.
[2] Big data meets big data analytics [Online]. Available: http://www.sas.com/content/dam/SAS/en_us/doc/whitepaper1/big-data-meets-big-data-analytics-105777.pdf, 2012.
[3] 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.
[4] W.-T. Balke, U. Güntzer, and J. X. Zheng, "Efficient distributed skylining for web information systems," in *Proc. 9th Int. Conf. Extending Database Technol.*, 2004, pp. 256–273.
[5] I. Bartolini, P. Ciaccia, and M. Patella, "Efficient sort-based skyline evaluation," *ACM Trans. Database Syst.*, vol. 33, no. 4, pp. 1–49, 2008.
[6] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel, "Finding a needle in haystack: Facebook's photo storage," in *Proc. 9th USENIX Conf. Operating Syst. Des. Implementation*, 2010, pp. 47–60.
[7] S. Börzsönyi, D. Kossmann, and K. Stocker, "The skyline operator," in *Proc. 17th Int. Conf. Data Eng.*, 2001, pp. 421–430.
[8] 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*, 2012, pp. 2262–2270.
[9] S.-R. Cho, J. Lee, S. won Hwang, H. Han, and S.-W. Lee, "VSkyline: Vectorization for efficient skyline computation," *ACM SIGMOD Rec.*, vol. 39, no. 2, pp. 19–26, 2010.
[10] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting," in *Proc. 19th Int. Conf., Data Eng.*, 2003, pp. 717–719.
[11] A. Cosgaya-Lozano, A. Rau-Chaplin, and N. Zeh, "Parallel computation of skyline queries," in *Proc. 21st Annu. Int. Symp. High Perform. Comput. Syst. Appl.*, 2007, p. 12.
[12] B. Cui, H. Lu, Q. Xu, L. Chen, Y. Dai, and Y. Zhou, "Parallel distributed processing of constrained skyline queries by filtering," in *Proc. IEEE 24th Int. Conf. Data Eng.* 2008, pp. 546–555.
[13] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.
[14] A. Eldawy, Y. Li, M. F. Mokbel, and R. Janardan, "CG_Hadoop: Computational geometry in MapReduce," in *Proc . 21st ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst.*, 2013, pp. 294–303.
[15] Facebook, Inc. [Online]. Available: http://www.sec.gov/Archives/edgar/data/1326801/000119312512175673/d287954ds1a.htm, 2012.
[16] Y. Gao, G. Chen, L. Chen, and C. Chen, "Parallelizing progressive computation for skyline queries in multi-disk environment," in *Proc. 17th Int. Conf. Database Expert Syst. Appl.*, 2006, pp. 697–706.
[17] P. Godfrey, R. Shipley, and J. Gryz, "Maximal vector computation in large data sets," in *Proc. 31st Int. Conf. Very Large Data Bases*, 2005, pp. 229–240.
[18] K. Hose and A. Vlachou, "A survey of skyline processing in highly distributed environments," *Int. J. Conf. Very Large Data Bases.*,vol. 21, no. 3, pp. 359–384, 2012.
[19] Z. Huang, C. S. Jensen, H. Lu, and B. C. Ooi, "Skyline queries against mobile lightweight devices in MANETs," in *Proc. 22nd Int. Conf. Data Eng.*, 2006, p. 66.
[20] H. Köhler, J. Yang, and X. Zhou, "Efficient parallel skyline processing using hyperplane projections," in *Proc. SIGMOD Int. Conf. Manage. Data.*, 2011, pp. 85–96.
[21] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: An online algorithm for skyline queries," in *28th Int. Conf. Very Large Data Bases*, 2002, pp. 275–286.
[22] H. T. Kung, F. Luccio, and F. P. Preparata, "On finding the maxima of a set of vectors," *J. ACM*, vol. 22, no. 4, pp. 469–476, 1975.
[23] K. C. K. Lee, B. Zheng, H. Li, and W.-C. Lee, "Approaching the skyline in Z order," in *Proc. 33rd Int. Conf. Very Large Data Bases*, 2007, pp. 279–290.
[24] 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, 2012.
[25] 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.
[26] K. Mullesgaard, J. L. Pedersen, H. Lu, and Y. Zhou, "Efficient skyline computation in MapReduce," in *Proc. Int. Conf. Extending Database Technol.: Adv. Database Technol.*, 2014, pp. 37–48.
[27] 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.
[28] 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.
[29] Y. Park, J.-K. Min, and K. Shim, "Parallel computation of skyline and reverse skyline queries using mapreduce," *Proc. VLDB Endowment*, vol. 6, no. 14, pp. 2002–2013, 2013.
[30] J. B. Rocha-Junior, A. Vlachou, C. Doulkeridis, and K. Nørvåg, "AGiDS: A grid-based strategy for distributed skyline query processing," in *Proc. 2nd Int. Conf. Globe*, 2009, pp. 12–23.
[31] C. Sheng and Y. Tao, "On finding skylines in external memory," in *Proc. 30th ACM SIGMOD-SIGACT-SIGART Symp. Principles Database Syst.*, 2011, pp. 107–116.
[32] 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.
[33] 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.
[34] S. Wang, B. C. Ooi, A. K. H. Tung, and L. Xu, "Efficient skyline query processing on peer-to-peer networks," in *Proc. IEEE 23rd Int. Conf. Data Eng.*, 2007, pp. 1126–1135
[35] P. Wu, C. Zhang, Y. Feng, B. Y. Zhao, D. Agrawal, and A. El Abbadi, "Parallelizing skyline queries for scalable distribution," in *Proc. 10th Int. Conf. Extending Database Technol.*, 2006, pp. 112–130.
[36] B. Zhang, S. Zhou, and J. Guan, "Adapting skyline computation to the MapReduce framework: Algorithms and experiments," in *Proc. 16th Int. Conf. Database Syst. Adv. Appl. Workshops*, 2011, pp. 403–414.
[37] 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.
[38] 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.

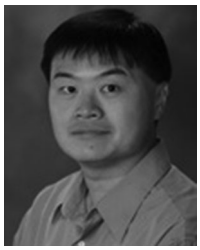
Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:33:12 UTC from IEEE Xplore. Restrictions apply.



Ji Zhang received the BS and MS degrees in computer science from Huazhong University of Science and Technology (HUST), Wuhan, China, in 2004 and 2007, respectively. He is currently working toward the PhD degree in the Department of Computer Science and Software Engineering, Auburn University. He was a software engineer in Huawei Technologies from 2007 to 2010. He is a member of the IEEE.



Xunfei Jiang received the BS and MS degrees in computer science from Huazhong University of Science and Technology (HUST), China, in 2004 and 2007, respectively. She received the PhD degree from the Department of Computer Science and Software Engineering, Auburn University in 2014. After receiving the MS degree, she joined Digital Video Networks Co., Ltd. in 2007 and Cisco Systems (Shanghai) Video Technology Co., Ltd. in 2010. She is currently an assistant professor in the Department of Computer Science, Earlham College. She is a member of the IEEE.



Wei-Shinn Ku (S'02-M'07-SM'12) received both the MS degree in computer science and the MS degree in electrical engineering in 2003 and 2006, respectively, from the University of Southern California (USC), where he also received the PhD degree in computer science in 2007. He is an associate professor with the Department of Computer Science and Software Engineering, Auburn University. He is a senior member of the IEEE.



Xiao Qin (S'00-M'04-SM'09) received the BS and MS degrees in computer science from the Huazhong University of Science and Technology, Wuhan, China, and the PhD degree in computer science from the University of Nebraska-Lincoln, Lincoln, in 1992, 1999, and 2004, respectively. For three years, he was an assistant professor with the New Mexico Institute of Mining and Technology, Socorro, NM. Currently, he is an associate professor with the Department of Computer Science and Software Engineering, Auburn University, Auburn, AL. He is a senior member of the IEEE.

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

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:33:12 UTC from IEEE Xplore. Restrictions apply.