Subarray Skyline Query Processing in Array Databases

Dalsu Choi
Korea University
Seoul, Korea
dalsuchoi@korea.ac.kr

Hyunsik Yoon
Korea University
Seoul, Korea
hyunsikyoon@korea.ac.kr

Yon Dohn Chung
Korea University
Seoul, Korea
ydchung@korea.ac.kr

ABSTRACT

With the generation of large-scale spatial data in various fields, array databases that represent space as an array have become one of the means of managing spatial data. Each cell in an array tends to interact with one another; therefore, instead of considering a single cell, considering a concept of subarray is required in some applications. In addition, each cell has several attribute values to indicate its features. Based on the two observations, we propose a new type of query, subarray skyline, that provides a way to find meaningful subarrays or filter less meaningful subarrays considering attributes. We also introduce an efficient query processing method, ReSKY, in centralized and distributed settings. Through extensive experiments using an array database and real datasets, we show that ReSKY has better performance than the existing techniques.

CCS CONCEPTS

• **Information systems** → **Database query processing**; *Parallel and distributed DBMSs*; *Data model extensions*.

KEYWORDS

subarray skyline, array databases, distributed processing

ACM Reference Format:

Dalsu Choi, Hyunsik Yoon, and Yon Dohn Chung. 2021. Subarray Skyline Query Processing in Array Databases. In *33rd International Conference on Scientific and Statistical Database Management (SSDBM 2021)*, July 6–7, 2021, Tampa, FL, USA. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3468791.3468799>

1 INTRODUCTION

Large-scale spatial data have been generated in various fields such as scientific domains and location-based services. Array databases have been steadily researched as a method to manage these spatial data [3, 29, 33, 40]. The array data model, the core of array databases, represents a given space as an array and a spatial object as a cell.

Each cell in an array tends to affect neighboring cells. In some cases, a set of neighboring cells is affected together by a change. These features raise the necessity to consider the concept of a subarray. In addition, each cell has a set of attributes that indicate its

characteristics. Therefore, it is required to consider subarrays with several attributes, as shown in the following real-world motivating scenarios.

Motivation 1 [Monitoring]. The Amazon rainforest performs important roles for the earth ecosystem. Unfortunately, the rainforest suffers from logging, fire, livestock farming, and so on in the vast expanse of land. Therefore, monitoring Amazon rainforest is a critical and fundamental mission. National Aeronautics and Space Administration (NASA) releases Normalized Difference Vegetation Index (NDVI) and Enhanced Vegetation Index (EVI) values at a regular time interval, where the higher NDVI and EVI values mean the higher density of vegetation on land [9]. Deforestation from infectious diseases or fire is usually observed not at a single cell but at a subarray because the diseases and fire influence the neighboring cells. The fundamental step is to "find subarrays in danger that no other subarrays have less NDVI and EVI values at the same time". The subarrays in danger include not only the subarray with the smallest NDVI value and the subarray with the smallest EVI value but also the subarrays in danger when considering both NDVI and EVI values. Based on the monitoring system, scientists figure out the candidate subarrays in danger regarding deforestation.

Motivation 2 [Decision Making]. Suppose a business manager plans to establish a branch of their company. It is important to select the most influential position to maximize their profits. One of the important factors is to consider living population and income of neighboring residences within the subarray that the candidate branch position can influence. Therefore, the process to select candidate positions can be seen as choosing the appropriate subarrays. In order to narrow the candidates, their requirement is described as "excluding subarrays that have less living population and income at the same time than another subarray" because the excluded subarrays are less attractive from their business model perspective. Then, the candidate subarrays become the input of the next level decision pipeline. The candidate subarrays at least guarantee that no other subarrays have more living population and income at the same time.

The above motivations can be mapped to the following problem definition; Given attribute values (such as NDVI, EVI, living population, and income) with their dimension values (such as latitude and longitude), a set of subarrays is returned as the output, where no other subarrays have the better attribute values at the same time than those of the output subarrays. In this definition, finding a set of tuples that no other tuples have better attribute values at the same time is known as a skyline query problem [4], which finds a Pareto-optimal subset of the multi-attribute dataset. In summary, we define the problems in the motivations as subarray skyline query processing in array databases.

We consider three aspects of the definition of subarray skyline. The first aspect is whether a subarray is the appropriate unit that

Corresponding author: Yon Dohn Chung

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SSDBM 2021, July 6–7, 2021, Tampa, FL, USA

© 2021 Association for Computing Machinery.

ACM ISBN 978-1-4503-8413-1/21/07...\$15.00

<https://doi.org/10.1145/3468791.3468799>

can represent the interactions between cells. A set of cells affected by a change may not form a rectangular subarray. We focus that any kind of shape can be easily bounded by a rectangle, that is, minimum bounding rectangles [15]. Although some detailed information can be lost, a rectangular subarray can represent an arbitrary shape in a reasonable manner. Second, in a single subarray skyline query, we need to consider whether the sizes of rectangular subarrays are fixed or variable. Various sizes of subarrays can enrich the output subarrays, but the computational cost increases significantly. In order to guarantee the reasonable response time for a query, we assume that the size of a subarray is fixed in a single query. Finally, we have to consider how to compare two subarrays. To support various user requirements, we assume all aggregate functions whose return value is a real number. In addition, a query can contain different aggregate functions on multiple attributes. For example, a single subarray in Motivation 2 can have two aggregated values, average living population and total income, computed from *average()* and *summation()* functions, respectively. In the rest of paper, we call the aggregated attribute values of a subarray the *representative*.

Figure 1: Subarray skyline results when a subarray size is 2×2 and the aggregate function is *average()* for both attributes.

Figure 1 shows a concrete example of the subarray skyline query processing. The two dimensions of the array consist of latitude and longitude, and the two attributes consist of living population and real estate price. Suppose that a subarray skyline query is given: (1) the fixed subarray size is 2×2 such as SA_1 and SA_2 in the figure, (2) the aggregate function is *average()* for both the living population and real estate price, and (3) a smaller value is preferred. By averaging the attribute values of each subarray, all non-empty subarrays that include at least one cell are mapped into the representatives, as shown in Figure 1(a). The subarray skyline consists of SA_1 and SA_2 . Without considering the concept of subarray, the conventional skyline consists of c_1 , c_2 , and c_3 , as shown in Figure 1(b). The subarray SA_2 , the conventional skyline query cannot see, has the most preferred average real estate price when considering the concept of subarray.

For efficient subarray skyline query processing, we consider the following research directions. The first research direction is to prune the search space efficiently because each subarray should be compared with a lot of different subarrays during the query processing. We form groups of subarrays based on the observation that the representatives of the neighboring subarrays tend to have similar values. We propose novel and efficient methods that identify dominance relationships among subarray groups. The methods prune non-skyline subarrays early and decrease unnecessary dominance checks among skyline subarrays. The second research direction is to support load-balanced distributed processing for scalability.

This is because the spatial dataset is usually large-scale, and the process requires expensive computational cost, which means that it sometimes cannot be handled in a single computing node. For the distributed processing, we propose group-based sampling methods in which each processing node can consider the subarrays of other nodes efficiently while decreasing the network cost and straggler effect.

Our contributions are summarized as follows. First, to the best of our knowledge, we propose a subarray skyline for the first time, which can provide novel insights into multi-attribute spatial data in array databases. Second, we propose an efficient subarray skyline query processing method, ReSKY, for both centralized and distributed environments. Finally, we carry out extensive experimental evaluations using real datasets and an array database.

The rest of this paper is organized as follows. Section 2 formally defines a subarray skyline query. In Section 3, we introduce the preliminaries before introducing our methods. Section 4 describes the main principles of ReSKY, and Section 5 introduces the detailed algorithms in a centralized environment. Section 6 extends ReSKY to distributed processing. Section 7 describes experimental evaluation. Finally, Section 8 discusses previous related studies, and Section 9 provides some concluding remarks.

2 PROBLEM DEFINITION

Definition 2.1 (Array). Suppose that a **dimension** D_i is a set of consecutive integers in $[L_i, U_i]$ and an **attribute** A_i is a set of real numbers. Given m dimensions $\{D_1, D_2, \dots, D_m\}$ and n attributes $\{A_1, A_2, \dots, A_n\}$, the **array** is defined as follows: $D_1 \times D_2 \times \dots \times D_m \mapsto (A_1, A_2, \dots, A_n)$. That is, a combination of dimension values defines the unique position of a set of attribute values in an array. $(d_1, d_2, \dots, d_m) \in D_1 \times D_2 \times \dots \times D_m$ is called a **cell**. A cell $(d_1, d_2, \dots, d_m)$ mapped to $\emptyset$ is called an empty cell. $\square$

For example, the array in Figure 1 has two dimensions, D_1 (latitude) and D_2 (longitude), and two attributes, A_1 (living population) and A_2 (real estate price). Cell $(0, 1)$ has $(16, 15)$ attribute values while cell $(0, 2)$ is an empty cell.

A **subarray** in an array is denoted by $SA(\text{starting cell, size})$, where the starting cell consists of the smallest dimension values in the subarray, and the size consists of the length of each dimension. If not confused, we briefly represent a subarray as $SA(\text{starting cell})$. A subarray where the whole cells are empty is called an empty subarray. For example, in Figure 1, a subarray, $SA((2, 0), 2 \times 2)$, consists of the four cells: cell $(2, 0)$, cell $(2, 1)$, cell $(3, 0)$, and cell $(3, 1)$. Another subarray, $SA((0, 2), 2 \times 2)$, is an empty subarray. Unless otherwise noted, we only consider non-empty subarrays in an array. Next, we introduce the basic dominance relationship among two tuples.

Definition 2.2 (Dominance). Given two tuples $T_1 = (t_{11}, t_{12}, \dots, t_{1k})$ and $T_2 = (t_{21}, t_{22}, \dots, t_{2k})$, T_1 dominates T_2 iff $\forall i \in [1, 2, \dots, k], t_{1i} \leq t_{2i}$ and $\exists i \in [1, 2, \dots, k], t_{1i} < t_{2i}$. If T_1 does not dominate T_2 , and T_2 does not dominate T_1 , then T_1 and T_2 are incomparable. $\square$

For example, if $T_1 = (1, 2, 2)$ and $T_2 = (1, 5, 2)$, T_1 dominates T_2 . If $T_1 = (3, 2, 2)$ and $T_2 = (1, 5, 2)$, they are incomparable. In the rest of the paper, we consider the *min* skyline in which smaller values are preferred, as shown in Definition 2.2. Our proposed method can be

easily extended to the *max* skyline. Now, we define the dominance relationship among two subarrays.

Definition 2.3 (Subarray Dominance). Suppose (1) an array with m dimensions and n attributes, (2) a subarray size, and (3) $k(\leq n)$ selected attributes $\{A'_1, A'_2, \dots, A'_k\}$ with corresponding aggregate functions $f = \{f_1, f_2, \dots, f_k\}$. The **representative of a subarray** SA is a tuple $SA.R = (SA.R_1, SA.R_2, \dots, SA.R_k)$ where $SA.R_i$ is f_i output value for all A'_i values in SA . A subarray SA_1 dominates another subarray SA_2 iff $SA_1.R$ dominates $SA_2.R$. Iff $SA_1.R$ and $SA_2.R$ are incomparable, then SA_1 and SA_2 are incomparable. $\square$

Note that for a subarray SA , the dimension values only define the unique position of SA in an array and do not participate in the subarray dominance, which is only based on attribute values. For example, for $SA((2, 0), 2 \times 2)$ in Figure 1, if $A'_1 = A_1, A'_2 = A_2$, and $f_1 = f_2 = \text{average}()$, then the representative $SA((2, 0), 2 \times 2).R$ is $(\text{average}(\{7, 8, 9\}), \text{average}(\{8, 7, 6\})) = (8, 7)$. Based on the definition of subarray dominance, we propose a subarray skyline query.

Definition 2.4 (Subarray Skyline). Given (1) an array, (2) the subarray size, and (3) the selected attributes with aggregate functions, the subarray skyline returns a set S of given-sized subarrays such that $\forall SA \in S$ is not dominated by any other given-sized subarrays in the array. $\square$

3 PRELIMINARIES

First, we introduce the space partitioning principle [47].

Definition 3.1 (Address of a Tuple). Given a tuple $T = (t_1, t_2, \dots, t_k)$ and a pivot $PV = (pv_1, pv_2, \dots, pv_k)$, the *address* of T regarding PV consists of k bits such that $\forall i \in [1, \dots, k]$, i^{th} bit is 0 if $t_i < pv_i$ and 1 if $pv_i \leq t_i$, which is denoted by $addr(T, PV)$ or simply by $addr(T)$ if not confused. $\square$

For example, if a tuple $T = (3, 5, 1)$, and a pivot $PV = (1, 6, 3)$, then $addr(T, PV) = 100$ which is a binary number with three bits. We introduce a relationship between dominance and *address*.

LEMMA 3.2. Given (1) two tuples $T_1 = (t_{11}, t_{12}, \dots, t_{1k})$ and $T_2 = (t_{21}, t_{22}, \dots, t_{2k})$, and (2) a pivot $PV = (pv_1, pv_2, \dots, pv_k)$, if T_1 dominates T_2 , $(addr(T_1, PV) \mid addr(T_2, PV)) = addr(T_2, PV)$. (" $\mid$ " denotes the bit-wise OR operator.)

PROOF. If $(addr(T_1, PV) \mid addr(T_2, PV)) \neq addr(T_2, PV)$, there exists at least one $i \in [1, 2, \dots, k]$ value, where the i^{th} bit of $addr(T_1, PV)$ is 1, and that of $addr(T_2, PV)$ is 0. This means that $t_{2i} < pv_i \leq t_{1i}$; thus, T_1 does not dominate T_2 by Definition 2.2. $\square$

For example, if $T_1 = (1, 2, 2)$, $T_2 = (1, 5, 2)$, and $PV = (0, 3, 4)$, then $addr(T_1, PV) = 100$, and $addr(T_2, PV) = 110$. Because T_1 dominates T_2 , $(addr(T_1, PV) \mid addr(T_2, PV)) = 110 = addr(T_2, PV)$.

The contraposition of Lemma 3.2 is that if $(addr(T_1) \mid addr(T_2)) \neq addr(T_2)$, then T_1 does not dominate T_2 ($= T_2$ is not dominated by T_1). That is, all tuples in $\{T_1 \mid (addr(T_1) \mid addr(T_2)) \neq addr(T_2)\}$ never dominates T_2 . From T_2 's perspective, when checking whether T_2 is dominated, T_2 does not have to be compared with all tuples in $\{T_1 \mid (addr(T_1) \mid addr(T_2)) \neq addr(T_2)\}$ because T_2 is never dominated by T_1 . This principle allows to prune unnecessary dominance checks.

4 THE MAIN PRINCIPLES OF ReSKY

We introduce the main principles of our proposed method, ReSKY. We assume that for a Minimum Bounding Rectangle (MBR) from a set of tuples, lower bound (MBR.LB) consists of the smallest value for each attribute, and upper bound (MBR.UB) consists of the largest value for each attribute among the tuples in the MBR. Note that the MBRs are defined over attribute value space, not dimension space. First, we define the two types of relationships among MBRs: dominance and partial dominance.

Definition 4.1 (MBR Dominance). Given two MBRs, $MBR1$ and $MBR2$, iff $MBR1.UB$ dominates $MBR2.LB$, $MBR1$ dominates $MBR2$, which means that any tuple in $MBR1$ dominates all tuples in $MBR2$. $\square$

The MBR dominance indicates that if $MBR1$ dominates $MBR2$, all the tuples in $MBR2$ can be discarded through one dominance check (between $MBR1.UB$ and $MBR2.LB$) without tuple-level dominance checks. Even if $MBR1$ does not dominate $MBR2$, there is another chance to prune tuple-level dominance checks from another loose relationship, called partial dominance.

Definition 4.2 (MBR Partial Dominance). Given two MBRs, $MBR1$ and $MBR2$, iff $MBR1.LB$ dominates $MBR2.UB$, $MBR1$ partially dominates $MBR2$, which means that some tuples in $MBR1$ may dominate tuples in $MBR2$. $\square$

The MBR partial dominance indicates that if $MBR1$ does not partially dominate $MBR2$, the dominance checks to identify whether a tuple in $MBR1$ dominates tuples in $MBR2$ are skipped through one dominance check (between $MBR1.LB$ and $MBR2.UB$) without tuple-level dominance checks. If $MBR1$ and $MBR2$ do not partially dominate each other, that is, two MBRs are incomparable, all the possible dominance checks between two MBRs are skipped. In light of the MBR partial dominance, a pair of MBRs are compared only when one MBR partially dominates another MBR.

Figure 2: Three types of relationships among MBRs

The problem is that identifying dominance/partial dominance among MBRs is also computational overhead as the number of MBRs increases, which seriously decreases the possible benefits from the MBR-level relationships. To address the problem, we introduce the MBR-level space partitioning principle.

OBSERVATION 1. Given (1) two MBRs, $MBR1$ and $MBR2$, and (2) a pivot PV , if $MBR1$ dominates $MBR2$, $(addr(MBR1.UB, PV) \mid addr(MBR2.LB, PV)) = addr(MBR2.LB, PV)$.

PROOF. $MBR1.UB$ dominates $MBR2.LB$ by Definition 4.1; therefore, $(addr(MBR1.UB, PV) \mid addr(MBR2.LB, PV)) = addr(MBR2.LB, PV)$ by Lemma 3.2. $\square$

The contraposition of Observation 1 is that if $(addr(MBR1.LB) \mid addr(MBR2.LB)) \neq addr(MBR2.LB)$, then $MBR1$ does not dominate $MBR2$. From $MBR2$'s perspective, when checking whether $MBR2$ is dominated, $MBR2$ does not have to be compared with all MBRs in $\{MBR1 \mid (addr(MBR1.LB) \mid addr(MBR2.LB)) \neq addr(MBR2.LB)\}$ because $MBR2$ is never dominated by $MBR1$.

OBSERVATION 2. Given (1) two MBRs, $MBR1$ and $MBR2$, and (2) a pivot PV , if $MBR1$ partially dominates $MBR2$, $(addr(MBR1.LB, PV) \mid addr(MBR2.LB, PV)) = addr(MBR2.LB, PV)$.

PROOF. $MBR1.LB$ dominates $MBR2.LB$ by Definition 4.2; therefore, $(addr(MBR1.LB, PV) \mid addr(MBR2.LB, PV)) = addr(MBR2.LB, PV)$ by Lemma 3.2. $\square$

The contraposition of Observation 2 is that if $(addr(MBR1.LB) \mid addr(MBR2.LB)) \neq addr(MBR2.LB)$, then $MBR1$ does not partially dominate $MBR2$. From $MBR2$'s perspective, when identifying the MBRs that partially dominates $MBR2$, $MBR2$ does not have to be compared with all MBRs in $\{MBR1 \mid (addr(MBR1.LB) \mid addr(MBR2.LB)) \neq addr(MBR2.LB)\}$.

Let us assume that a set of MBRs is given with a pivot. First, we calculate the addresses of LB s and UB s of the MBRs and divide them into **subsets** according to their addresses, where each subset consists of LB s and UB s with the same address. Then, we define the dominating subsets of a subset.

Definition 4.3 (Dominating Subsets). Given a pivot PV , dominating subsets of a subset S regarding PV , **dominating subsets**(S, PV), are $\{S' \mid (addr(S') \mid addr(S)) = addr(S)\}$, where $addr(S)$ is the address of the LB s and UB s in S regarding PV . $\square$

When checking whether a MBR is dominated by another MBR, Observation 1 indicates that it is enough for $MBR.LB$ to be compared only with the selected UB s, $\{UB \mid (addr(UB) \mid addr(MBR.LB)) = addr(MBR.LB)\}$, that may dominate $MBR.LB$. In subset-level, it is enough for LB s in a subset S to be compared with UB s only in its dominating subsets. When identifying the MBRs which partially dominates MBR , Observation 2 indicates that it is enough for $MBR.LB$ to be compared only with the selected UB s, $\{UB \mid (addr(UB) \mid addr(MBR.LB)) = addr(MBR.LB)\}$, that may dominate $MBR.LB$. In subset-level, it is enough for UB s in a subset S to be compared with LB s only in its dominating subsets. Note that a subset itself is in its dominating subsets.

In summary, the main principle of ReSKY integrate the advantage of MBR-level dominance/partial dominance check and the space partitioning principle. That is, we can prune non-skyline tuples on a MBR basis (from MBR dominance) and unnecessary tuple-level dominance checks among incomparable tuples (from MBR partial dominance) while decreasing the computational overhead from MBR-level dominance checks (from MBR-level space partitioning).

5 THE ALGORITHMS OF ReSKY

Now, we explain the algorithms of ReSKY. ReSKY in a centralized environment consists of three steps: (1) forming subarray groups, (2) subarray group skyline computation, and (3) subarray skyline computation. In Section 4, we introduced the terms, **tuple** and **MBR**. In the discussion that follows in Section 5, we replace the notion of **tuple** with that of **representative of a subarray**, and the notion of **MBR** with that of **MBR of a subarray group**.

5.1 Subarray Grouping

The first step of ReSKY for processing a subarray skyline query is to compute representatives of all subarrays in the array. In order to apply the main principle in Section 4, we need MBRs of the subarrays. Based on the observation that the representatives of dimension-adjacent subarrays tend to have similar attribute values, we form subarray groups and extract their MBRs.

Definition 5.1 (Subarray Group, SG). Given (1) fixed-sized subarrays in an array with m dimensions and (2) a subarray group size $\{g_1, \dots, g_m\}$, a subarray group $SG((d_1, \dots, d_m))$ consists of a set of subarrays, $SG.subarrays$, where for each starting cell $(s_1, \dots, s_m)$, $d_i \leq s_i \leq d_i + g_i - 1$, for all $i \in [1, \dots, m]$. $\square$

The MBR of a subarray group SG , $MBR(SG)$, is calculated from the representatives of the subarrays in the group. Note that the representative of a subarray is aggregated from attribute values whereas the dimension values define the unique position of the subarray in the array. $MBR(SG).LB$ and $MBR(SG).UB$ are simply denoted as $SG.LB$ and $SG.UB$, respectively. In addition, when $MBR(SG1)$ (partially) dominates $MBR(SG2)$, we denote that $SG1$ (partially) dominates $SG2$.

Figure 3: Subarray groups from the array in Figure 1 (the subarray size is 2×2 , the aggregate function is *average()* for both attributes, and the group size is $\{1, 2\}$).

Figure 3 shows the example subarray groups from the array in Figure 1. We form subarray groups such that a subarray is included in only one subarray group. The grouping starts with the starting cell of the array, that is, the cell with the smallest dimension values for all dimensions. When the current subarray group is $SG((d_1, d_2, \dots, d_m))$, the next subarray group is $SG((d'_1, d'_2, \dots, d'_m)) = SG((d_1, d_2, \dots, d_{m-1}, d_m + g_m))$. If $d_m + g_m$ is out of the possible dimension range in the array, then $d_m + g_m$ is set to the smallest value for the dimension, and d_{m-1} is updated to $d_{m-1} + g_{m-1}$, and this carry process is the same for other dimensions. When the carry happens at the first dimension, the grouping process is done. $SG((0, 2))$ does not include $SA((0, 2))$ because it is empty.

The grouping process is executed at query runtime because the subarray size and aggregate functions are not known before the runtime. However, this process enables grouping to be completed in $O(xn)$, where x is the total number of subarrays, and n is the number of attributes, which is much more efficient than bulk-loading an R-tree's leaf nodes from all subarrays to obtain the MBRs.

5.2 Subarray Group Skyline

After getting subarray groups, the second step of ReSKY is to find a set of subarray groups, **SG-Skyline**, such that the MBR of a subarray group in the set is not dominated by the MBRs of any other subarray groups based on Definition 4.1 and Observation 1. Therefore, the primary purpose of the second step is to prune a part of non-skyline subarrays only based on MBRs without checking the subarrays in each MBR. The number of MBRs is much

smaller than that of the subarrays by the factor of the subarray group size; therefore, computing SG-Skyline can be performed with less computational cost than subarray-level skyline computation. A naïve method used to find SG-Skyline is to check all possible pairs of MBRs, which becomes time-consuming as the number of MBRs increases in large-scale data although the number of MBRs is smaller than that of subarrays. Instead, it is enough for a MBR to be compared with its dominating subsets as described in Section 4.

Algorithms 1 and 2 describe how to find SG-Skyline. The main idea is to check only dominating subsets for each subset. Given a set of *LBs* and *UBs* of SGs, Algorithm 1 selects the median as a pivot to partition the set in a balanced manner (Line 2), calculates their *addresses* (Definition 3.1), and divide the *LBs* and *UBs* into subsets (Line 3). The algorithm discards *LBs* in the subset with *address* 11...1 if there is a *UB* in the subset with *address* 00...0 without any further comparison. From the subset with the smallest *address*, the algorithm checks whether *LBs* in the subset are dominated by a *UB* in its dominating subsets excluding itself (Line 5). After checking the dominating subsets, the algorithm decides whether to recursively partition the current subset (Lines 6–11). Because Algorithm 1 is a recursive function, the dominating subsets might be further divided; Algorithm 2 handles the case with the same principle.

Figure 4: Finding SG-Skyline from SGs

Figure 4 shows an example of finding SG-Skyline among the MBRs of subarray groups. Algorithm 1 first computes the median $PV1 = (7.5, 6.5)$ of $LBs = UBs = \{SG1, SG2, SG3, SG4, SG5, SG6\}$, and divides them according to their *addresses* regarding $PV1$. Because $addr(SG3.UB, PV1)$ is 00, SG6 is discarded without further comparison. The algorithm processes the subset, S00, with 00 *address*, where $S00.LBs = \{SG2, SG3, SG5\}$ and $S00.UBs = \{SG3\}$. Because $S00.LBs$ does not have a dominating subset except for itself, and $|S00.LBs| > 1$, S00 is recursively divided. $PV2 = (5.5, 2.5)$ is the median of $SG2.LB, SG3.LB, SG3.UB$, and $SG5.LB$. SG3, SG5, and SG2 become the new members of SG-Skyline. The algorithm processes the next subset, S01, with 01 *address*, where $S01.LBs = \{SG1\}$ and $S01.UBs = \{SG1\}$. Because S01 has S00 as the dominating subset, the algorithm calls Algorithm 2 and comes to know that SG1 is not dominated by all *UBs* in S00. SG1 becomes a new member of SG-Skyline. The next subset S10 consists of $S10.LBs = \{SG4\}$ and $S10.UBs = \{SG2\}$. The dominating subsets of S10 except for itself is S00 excluding S01. $SG4.LB$ is dominated by $SG3.UB$ in S00_S10. S11 does not have *LB*; therefore, the search for SG-Skyline terminates, as shown in Figure 4(b).

Algorithms 1 and 2 ensure that a subarray group SG, which is not a member of SG-Skyline, is dominated by another subarray group because the algorithms check all other subarray groups $\{SG' | (addr(SG'.UB) | addr(SG.LB)) = addr(SG.LB)\}$ until SG is dominated. If x is the number of SGs, k is the number of attributes, and the probability that an SG is dominated after a dominance check is p , then the time complexity of Algorithm 1 is $O(\frac{kx \log x}{p})$.

Algorithm 1: SG_SKY

```

Input : a set of LBs and UBs
Output : SG-Skyline
1 SG-Skyline  $\leftarrow \emptyset$ ;
2 Select median of all LBs and UBs in the set as a pivot;
3 Calculate addresses of LBs and UBs regarding the pivot, and
  divide them into subsets according to their addresses;
4 foreach  $S \in$  subsets in ascending order of addresses do
5   foreach  $LB \in S$  (and its corresponding SG) and
     foreach  $S' \in$  dominating subsets( $S, pivot$ )- $S$ ,
       if  $CHECK\_SET_{SG}(LB, S') = true$ ,
         delete  $LB$  from  $S$  and  $SG$ .
6   if the number of LBs in  $S \leq 1$  then
7     Check whether  $LB$  is dominated by a UB in  $S$ ;
8     SG-Skyline  $\leftarrow SG-Skyline \cup \{LB's\ SG\}$ ;
9   else
10    SG-Skyline  $\leftarrow SG-Skyline \cup SG\_SKY(S)$ ;
11  end
12 end
13 return SG-Skyline;

```

Algorithm 2: CHECK_SET_{SG}

```

Input :  $LB$ , a set  $S$  of LBs and UBs
Output : true if  $LB$  is dominated, else false
1 if  $S$  is not divided anymore then
2   if  $LB$  is dominated by a UB in  $S$ , return true;
3 else
4   Calculate the subset  $S_{LB}$  of  $LB$  regarding the pivot of  $S$ ;
5   foreach  $S' \in$  dominating subsets( $S_{LB}, pivot$  of  $S$ ) do
6     if  $CHECK\_SET_{SG}(LB, S')$ , return true;
7   end
8 end
9 return false;

```

5.3 Subarray Skyline

The final step of ReSKY is to compute a set of subarrays, **Subarray-Skyline**, that each subarray in the set is not dominated by any other subarrays when given SG-Skyline. Even if some subarray groups are pruned in the second step, comparing a subarray with all the remaining subarrays can be computationally expensive. To further prune the dominance checks, we utilize the principle of Definition 4.2 and Observation 2. In other words, it is enough for the subarrays in a MBR to be compared with subarrays in its dominating subsets as described in Section 4.

Given SG-Skyline from Algorithm 1, Algorithm 3 describes how to determine Subarray-Skyline. Algorithm 3 has a structure similar to that of Algorithm 1. The differences are as follows: First,

because the given subarray groups are SG-Skyline members, the algorithm selects a pivot without considering pruned subarray groups. Second, for a subset S , the algorithm considers UBs in S and LBs in its dominating subsets, whereas Algorithm 1 considers LBs in S and UBs in its dominating subsets (Line 5 in Algorithm 3 and Line 2 in Algorithm 4). Third, given two subarray groups SG and SG' , if SG is partially dominated by SG' , Algorithms 3 and 4 check whether each subarray in $SG.subarrays$ is dominated by a subarray in $SG'.subarrays$. Note that if the representative of $SA \in SG.subarrays$ is not dominated by $SG'.LB$, SA is not dominated by all subarrays in $SG'.subarrays$; in this case, the algorithms immediately skip checking SA , which reduces unnecessary comparison among subarrays (Figure 6, case 3). Fourth, the algorithm determines the recursive call not by the size of LBs but by the size of UBs (Lines 6-12, Algorithm 3). Finally, after the subarrays of a subarray group are decided as the new members of Subarray-Skyline, if the number of subarrays in the group is changed compared to the original number, the algorithm recalculates the MBR of the group (Line 9, Algorithm 3). The MBR update can help to decrease cases that unchecked subarray groups are partially dominated by the subarray group.

Figure 5: Finding Subarray-Skyline from SG-Skyline

Figure 5 shows an example of finding Subarray-Skyline from SG-Skyline in Figure 4. We assume that each MBR has two subarrays in two vertices that are not LB and UB . Algorithm 3 calculates the median PV of $LBs = UBs = \{SG1, SG2, SG3, SG5\}$ and partitions them according to their *addresses* concerning PV . Because the subset $S00$ with 00 address does not have a UB , the algorithm moves to the next subset $S01$, where $S01.LBs = \{SG1, SG5\}$, and $S01.UBs = \{SG1\}$. The algorithm finds that $SG1$ is partially dominated by $SG3$, eliminates the subarray dominated by a subarray in $SG3$, inserts $SG1.subarrays$ into Subarray-Skyline, and updates $MBR(SG1)$. In the next subset $S10$, $SG2$ is partially dominated by $SG3$, and $SG2.subarrays$ are all dominated by the subarray whose representative is $(4, 2)$ in $SG3.subarrays$. $S10$ does not have to be compared with $S01$ because $S01$ is not a dominating subset of $S10$. Here, $SG3$ is partially dominated by itself, and $SG3.subarrays$ becomes new members of Subarray-Skyline. $SG5$ in $S11$ is partially dominated by $SG3$, and $SG5.subarrays$ are all dominated.

Algorithms 3 and 4 are correct because they ensure that each subarray group SG is checked with all subarray groups that partially dominates SG . If x is the number of SGs , k is the number of attributes, the maximal number of subarrays in a SG is $|sub|$, and the

Figure 6: Three cases under partial dominance relationship
maximal number of SGs that partially dominates a subarray group is p , then the time complexity of Algorithm 3 is $O(kpx(\log x + |sub|^2))$.

Algorithm 3: SUB_SKY

Input : a set of LBs and UBs
Output : Subarray-Skyline

- 1 Subarray-Skyline $\leftarrow \emptyset$;
- 2 Select median of all LBs and UBs in the set as a pivot;
- 3 Calculate *addresses* of LBs and UBs regarding the pivot, and divide them into subsets according to their *addresses*;
- 4 **foreach** $S \in$ subsets in ascending order of *addresses* **do**
- 5 **foreach** $UB \in S$ (and its corresponding SG) **and**
- 6 **foreach** $S' \in$ dominating subsets($S, pivot$)- S ,
 if $CHECK_SET_SUB(SG, S') = true$,
 delete UB from S and SG .
- 7 **if** the number of UBs in $S \leq 1$ **then**
- 8 For the subarray group SG of UB and each subarray group SG' that partially dominates SG where $SG'.LB \in S$, check whether $SG.subarrays$ are dominated by $SG'.subarrays$;
- 9 Subarray-Skyline $\leftarrow$ Subarray-Skyline $\cup$ $SG.subarrays$;
- 10 recalculate $MBR(SG)$ if it may shrink;
- 11 **else**
- 12 Subarray-Skyline $\leftarrow$ Subarray-Skyline $\cup$ SUB_SKY(S);
- 13 **end**
- 14 **return** Subarray-Skyline;

Algorithm 4: CHECK_SET_SUB

Input : SG , a set S of LBs and UBs
Output : $true$ if $|SG.subarrays| = \emptyset$; else, $false$

- 1 **if** S is not divided anymore **then**
- 2 **foreach** subarray group SG' that $SG'.LB \in S$ dominates $SG.UB$ ($= SG'$ partially dominates SG), check whether $SG.subarrays$ are dominated by $SG'.subarrays$. If $|SG.subarrays| = \emptyset$, return $true$;
- 3 **else**
- 4 Calculate the subset $S_{SG.UB}$ of $SG.UB$ regarding the pivot of S ;
- 5 **foreach** $S' \in$ dominating subsets($S_{SG.UB}, pivot$ of S) **do**
- 6 **if** $CHECK_SET_SUB(SG, S') = true$;
- 7 **end**
- 8 **end**
- 9 **return** $false$;

6 DISTRIBUTED PROCESSING

6.1 Chunking Strategy of Array Databases

Before extending ReSKY to distributed processing, we first introduce the chunking strategy widely used in array databases [33]. To handle a large-scale array, the array should be divided into smaller units called **chunks**, which become I/O and network transfer units. An array is divided into fixed-sized chunks based on the dimensions. Because an array has several attributes, a chunk is further divided according to the attributes, which means a column store. We call the set of chunks that have the same dimension values a **chunkset**, which is stored on the same processing node. To process a boundary subarray that overlaps with more than two chunks, each chunk has an appropriate size of **chunk overlaps**. If a chunk does not have enough chunk overlaps and the required chunks to process a boundary subarray are in different nodes, network transfer is required, which seriously decreases the query performance. Specifically, to compute a $|Ds_1| \times \dots \times |Ds_m|$ -sized boundary subarray, $\forall i \in [1, 2, \dots, m]$, at least $\lceil (|Ds_i| - 1)/2 \rceil$ -sized chunk overlaps are required.

Figure 7: An array is divided into 4×4 -sized chunks with $\{1, 1\}$ -sized chunk overlaps. Four chunksets and 12 chunks are distributed over two nodes.

6.2 Distributed Processing of ReSKY

Based on the chunking strategy, representative computation can be easily scaled out because each node just checks their subarrays without communicating with other nodes. However, we need solutions for the possible straggler problem of ReSKY in a distributed setting. Without loss of generality, we assume that (1) a database system consists of a single coordinator node (coordinator) and multiple worker nodes (workers), (2) all chunks with appropriate chunk overlaps in an array are stored on multiple workers without duplicate chunks, and (3) all workers have an equal number of non-empty chunksets that have at least one non-empty cell.

When a subarray skyline query arrives, the **first step (S1)** is to compute representatives of all the subarrays in the array and form the subarray groups on a chunkset basis. Each worker can compute the representatives without duplicate subarrays among different workers in parallel. In the ideal case, when the number of workers increases by x times, the representative computation time decreases by x times.

We pursue the following objective for load-balanced distributed processing [**Subarray-level objective, SA objective**].

$$\begin{aligned} \text{maximize } \{ \text{score} = & \sum_{i=1}^{|workers|} \frac{|global\ skyline \cap local\ skyline_i|}{|local\ skyline_i|} \} \\ \text{such that } \forall i, j \in & workers, |local\ skyline_i| = |local\ skyline_j| \\ & \text{and } |original\ subarrays_i| = |original\ subarrays_j| \end{aligned} \quad (1)$$

Distributed processing steps of ReSKY

- Step1 (S1):** Each worker computes representatives of its own subarrays and forms the subarray groups.
- Step2 (S2):** Each worker finds local SG-Skyline.
- Step3 (S3):** The coordinator computes global sample SG-Skyline, which enables each worker to prune its own SGs.
- Step4 (S4):** Each worker finds local Subarray-Skyline.
- Step5 (S5):** The coordinator computes global sample Subarray-Skyline, which enables each worker to prune its own subarrays.
- Step6 (S6):** The coordinator gathers local Subarray-Skylines.
- Step7 (S7):** The coordinator determines global Subarray-Skyline.

Ideally, when achieving SA objective, (1) each local skyline consists of only the subset of the global skyline, (2) every worker contributes equally to the global skyline, (3) the times for workers to compute the local skyline are similar, and (4) $\text{score} = |workers|$. However, realizing the ideal case when subarrays are already distributed among workers is not easy because (1) precisely estimating the size of the skyline is not trivial because the size is data-dependent, and (2) for each worker, it is difficult to know whether its local skyline will be part of the global skyline. In other words, we should know the global skyline in advance to achieve the ideal case of SA objective.

To overcome this issue, SA objective can be projected to the following objective [**SG-level objective, SG objective**].

$$\begin{aligned} \text{maximize } \sum_{i=1}^{|workers|} & \frac{|global\ SG-Skyline \cap local\ SG-Skyline_i|}{|local\ SG-Skyline_i|} \\ \text{such that } \forall i \in & workers, |local\ SG-Skyline_i| \geq \alpha \end{aligned} \quad (2)$$

Ideally, when achieving SG objective, (1) each worker outputs only the subarrays included in global SG-Skyline, (2) every worker contributes over a certain amount to the global skyline, and then (3) ReSKY reaches closer to SA objective.

In the **second step (S2)**, each worker finds **local SG-Skyline** independently using Algorithm 1. A naïve approach to achieve SG objective is that the coordinator gathers MBRs of local SG-Skylines without subarrays from all workers, computes global SG-Skyline, and makes every worker have an equal number of parts of global SG-Skyline without duplicates. However, when the number of SGs becomes large, the coordinator becomes a straggler while computing global SG-Skyline, although the number of SGs is usually much smaller than the number of subarrays. Instead, to reach closer to SG objective, we propose a sampling-based approach that minimizes the straggler problem, which is the **third step (S3)**: (1) Each worker randomly samples $sr\%$ SGs from its own local SG-Skyline. (2) The coordinator gathers MBRs of the sampled SGs without subarrays and computes **global sample SG-Skyline** using Algorithm 1. During the computation, SGs from the same worker do not have to be compared. (3) The coordinator broadcasts global sample SG-Skyline to all workers. (4) Each worker tries to prune its own SGs in local SG-Skyline using global sample SG-Skyline. Note that the pruning process is conducted efficiently based on the pivots found during the second step. (5) If $|local\ SG-Skyline_i|$ in worker i is below α , it receives random SGs with subarrays from another worker that has the largest number of SGs until worker i has α subarrays. The third step is meaningful in the sense that (1) computational overhead of

the coordinator reduces because the coordinator only considers the samples, and (2) each worker can consider meaningful SGs of other workers while minimizing network overhead. Each worker prunes all SGs dominated by global sample SG-Skyline; therefore, the remaining SGs can be seen as reaching closer to global SG-Skyline than before the third step. The communication cost for the third step is proportional to $(\sum_{i=1}^{|workers|} |MBR(SG)s \text{ of local SG-Skyline}_i| \times \frac{sr}{100}) + |MBR(SG)s \text{ of global sample SG-Skyline}| \times |workers|$ if S3-(5) is not needed.

In the **fourth step (S4)**, each worker finds **local Subarray-Skyline** independently using Algorithm 3 from the updated local SG-Skyline. Only SGs that contain at least one local subarray skyline remain.

Because a skyline operator is distributive [14], the union of local Subarray-Skylines always contains **global Subarray-Skyline**. Before merging local Subarray-Skylines, to reach closer to SA objective by pruning subarrays in local Subarray-Skylines, we re-sample SGs from local Subarray-Skyline, as the **fifth step (S5)**: (1) Each worker randomly samples $sr\%$ SGs from its own local Subarray-Skyline. (2) The coordinator gathers the sampled SGs with their subarrays and computes **global sample Subarray-Skyline** using Algorithm 3. The algorithm can skip the comparison between SGs from the same worker. (3) The coordinator broadcasts global sample Subarray-Skyline to all workers along with the MBR(SG)s. (4) Each worker tries to prune its own subarrays in local Subarray-Skyline using global sample Subarray-Skyline. Note that the pivots found in the fourth step is utilized to accelerate the pruning. The communication cost for the fifth step is proportional to $(\sum_{i=1}^{|workers|} |MBR(SG)s \text{ of local Subarray-Skyline}_i| \times \frac{sr}{100} \times |maximal \text{ number of subarrays in a SG}|) + |global \text{ sample Subarray-Skyline}| \times |workers|$. Through the fifth step, (1) ReSKY approaches SA objective, (2) when the size of local Subarray-Skyline decreases, network overhead required to send local Subarray-Skyline to the coordinator in the sixth step also decreases, and (3) computational overhead that occurs when the coordinator computes global Subarray-Skyline in the seventh step reduces.

In the **sixth step (S6)**, the coordinator gathers updated local Subarray-Skylines from all workers along with their MBR(SG)s. In the **seventh step (S7)**, the coordinator determines global Subarray-Skyline using Algorithm 3. Algorithm 3 can find Subarray-Skyline, even if the given SGs are not in SG-Skyline. The computation conducted during S4 and S5 are not repeated in S7.

Before wrapping up the section, we briefly discuss how to determine the parameters of ReSKY: a group size and sampling rates. The best group size and sampling rates depend on attribute value distributions, subarray size, selected attributes, and aggregate functions; obtaining the best parameters without running the query is not trivial. Therefore, we propose randomly choosing a subset of chunksets, testing several parameters, and choosing the best parameters among the candidates.

7 EXPERIMENTAL EVALUATION

7.1 Experimental Setup

Settings. We conducted our experiments using a server cluster with eight nodes, each of which consisted of an Intel(R) Core(TM)

i5-9400F CPU (2.90GHz), 64 GB memory, and a 2 TB hard disk. The server cluster was configured based on Ubuntu diskless booting (Ubuntu 14.04), and each node ran SciDB Community Edition 19.3 [33]. The first node served as both the coordinator and worker. The coordinator was connected to all workers through a 1000Mbps network switch. All evaluated methods were implemented as user-defined operators of SciDB using C++14.

Evaluated Algorithms. The algorithms that we implemented for comparison are as follows. The parameter values in the below were chosen after careful parameter tuning.

- (1) **ReSKY**: Our proposed methods. Reservoir sampling [37] was used for S3 and S5 in distributed processing.
- (2) **OSPS** [47]: Space partitioning-based method. The method selects a random skyline as a pivot. We implemented *OSPSOnPartitioningFirst*.
- (3) **VMPSP** [45]: Space partitioning-based method. The method selects the median of the given subarrays as a pivot. We set β to 5.
- (4) **SKY-SB** [44]: R-tree index-based method. Instead of bulk-loading an R-tree at query runtime, we applied *I-SKY^{DS}* and *E-DG-1* after forming subarray groups as in ReSKY.
- (5) **ZSearch** [24]: Index-based method. After representative computation at query runtime, we first converted double-type attributes into integer-type attributes to apply the Z-order curve property and build ZBtree. Then, ZSearch was applied. We set N to 100.
- (6) **AGiDS** [31] + **VMPSP/OSPS**: Grid-based distributed processing method based on the coordinator-worker model. We set the number of grids to 59,049.
- (7) **Shuffle**: After representative computation, the coordinator gathered all subarrays from all workers, randomly divided the entire subarrays into disjoint subsets with an equal size, and sent each subset back to each worker without any operations. Shuffle served as a guideline for distributed methods based on how to partition data [35, 38].
- (8) **SKY-MR-S/SKY-MR-M** [30] + **VMPSP**: One of the state-of-the-art MapReduce-based skyline query processing methods. We set $s = 400$ and $\rho = 20$.

All the evaluated algorithms require representative computation. After the computation, each subarray is mapped to a tuple, i.e., the representative. The existing skyline algorithms can be applied to the set of tuples to find Subarray-Skyline.

Datasets and Queries. [**NASA MODIS Dataset**] We used MOD09A1 [36] and MOD13Q1 [9]. MOD09A1 consists of 2400×2400 arrays with 13 attributes. MOD13Q1 consists of 4800×4800 arrays with 12 attributes. These datasets are generated at regular time intervals; therefore, we can select different data by selecting different times. In the single-node experiments, we randomly selected 15 MOD09A1 and 15 MOD13Q1 data (total 30 different data), and set the subarray size to 50×50 . In the multiple-node experiments, we appropriately merged data at different times to increase the size. For example, we merged 54 MOD09A1 data to form a $54 \times 2400 \times 2400$ array with approximately 3×10^8 subarrays. We set the subarray size to $1 \times 50 \times 50$.

[**Real Estate Dataset**] We crawled real estate information from one of the most popular online real estate agent services in Korea (<https://www.zigbang.com/>). Specifically, we extracted 2 dimensions (latitude and longitude of each apartment complex) and 9 attributes (lease/selling price per $3.3m^2$, lease/selling price and

total area for each residence in an apartment, the number of months since the complex was built, the distance from the nearest elementary/high school, and the number of residences in the apartment complex). The number of apartment complexes around Seoul ($132km \times 94km (= 7152 \times 6427 \text{ array})$) was 10,400. We used $1.6km \times 1.3km (= 90 \times 90)$ subarray size, which resulted in 8,505,179 unique subarrays.

[Restaurant Dataset] We crawled restaurant information from the most popular portal service in Korea (<https://www.naver.com/>). Specifically, we extracted 2 dimensions (latitude and longitude of each restaurant) and 6 attributes (the rating, the number of visitor reviews, the number of blog reviews, the price of the cheapest/most expensive menu, and the average price of menus). The number of popular restaurants around Seoul ($134km \times 95km (= 4797 \times 4269 \text{ array})$) was 37,597. We used $1.4km \times 1.1km (= 50 \times 50)$ subarray size, which resulted in 6,329,634 unique subarrays.

We set the chunk size to 500×500 (two dimensional arrays) or $1 \times 500 \times 500$ (three dimensional arrays) with appropriate chunk overlaps. SciDB distributes the chunks in an array to the workers with a hash function, which is a default setting. We used *average()* to compute the representatives of subarrays. The Subarray-Skyline ratio in the graphs indicates $|\text{Subarray-Skyline}|/|\text{total subarrays}| \times 100$. The Subarray-Skyline ratio implies the distribution of the dataset. If the ratio is high, a lot of subarrays belong to Subarray-Skyline, which means that the distribution is close to anti-correlated. If the ratio is low, the distribution is close to correlated [4]. *rep comp* indicates the representative computation time. We used K for 10^3 and M for 10^6 . For ReSKY, unless otherwise noted, we set the subarray group size to $\{10, 10\}$ in the single-node experiments and $\{1, 10, 10\}$ in the multiple-node experiments. The default sampling rate for S3 and S5 was 10. α in S3 was set to $\{(\sum_{i=1}^{|workers|} |\text{localSG-Skyline}_i \text{ after S3-(4)}|)/|workers|\} \times 0.5$. All the data structures required to process a query were loaded on main memory.

7.2 Experimental Results

We first conducted the single-node experiments to compare ReSKY with the existing skyline query processing algorithms, as shown in Figures 8 and 9. All numbers and execution times are the average values of 15 data. The execution times are shown on a log scale. Because all the evaluated algorithms require representative computation, we excluded *rep comp* to clearly compare the performance of the skyline query processing; ReSKY and SKY-SB included subarray grouping times. Figure 8 shows the results according to the number of subarrays when the number of attributes was 10. Figure 9 shows the results according to the number of attributes when the number of subarrays was 1000K. As shown, ReSKY outperformed the existing methods. As the fundamental reasons for such performance, ReSKY (1) pruned the non-skyline subarrays at SG-level, (2) decreased unnecessary dominance checks among incomparable subarrays by identifying partial dominance efficiently, and (3) found such dominance relationships among SGs efficiently. ReSKY outperformed SKY-SB because SKY-SB required more dominance checks to identify the relationships among *MBR(SG)*s, particularly when the number of *MBR(SG)*s became large. ZSearch suffered from ZBtree maintenance overhead caused by the tree node splits.

(a) MOD09A1 (average of 15 data) (b) MOD13Q1 (average of 15 data)

Figure 8: Execution times according to the number of subarrays (10 attributes, 1 worker).

(a) MOD09A1 (average of 15 data) (b) MOD13Q1 (average of 15 data)

Figure 9: Execution times according to the number of attributes (1000K subarrays, 1 worker).

In Figure 8, we observed that OSPS and VMPSP terminated faster than SKY-SB and ZSearch in the real datasets. Figure 9 shows that when the number of attributes was small, most of the methods performed well; however, the execution times of the methods except for ReSKY increased faster than ReSKY as the number of attributes increased. When the Subarray-Skyline ratio was small, OSPS terminated faster than VMPSP because selecting a random skyline as a pivot and quickly pruning non-skyline subarrays were better than trying to choose a good pivot. However, as the Subarray-Skyline ratio increased, it became more important to select a pivot that partitioned the subarrays in a balanced manner.

Figure 10 shows the execution times for each step of ReSKY according to various subarray group sizes. All numbers and execution times are the average values of 15 data. As the group size increased, the number of SGs decreased, which means that the time to form the subarray groups and compute SG-Skyline also decreased. By contrast, when computing Subarray-Skyline, a different pattern was shown in that an inefficiency occurred when the group size was too small or large. When the group size was too large, the number of common cells among the subarrays in a SG decreased, which means that the representatives of the subarrays tended to be dissimilar and *MBR(SG)*s enlarged; even if the number of SGs decreased, the number of dominance checks at subarray-level increased. When the group size was too small, although the subarrays with similar representatives were more likely to form SGs, identifying the partial dominance became overhead due to the large number of SGs. We also experimented our parameter tuning method introduced at the end of Section 6.2. Specifically, we randomly selected one chunkset at each data and tested various group sizes. The tuning took average 7.869 seconds for each data. The result indicates that the tuning method can provide a useful guideline.

(a) MOD09A1 (average of 15 data) (b) MOD13Q1 (average of 15 data)
Figure 10: Execution times according to different subarray group sizes (10 attributes, 1 worker).

Figure 11: Execution times according to the number of subarrays (10 attributes, 8 workers).

Figure 12: Execution times according to the number of attributes (25M subarrays, 8 workers).

We conducted experiments on distributed processing; Figure 11 shows the execution times according to the number of subarrays for 10 attributes, and Figure 12 shows the results according to the number of attributes when the number of subarrays was 25M. The experiments were conducted on eight workers. Because all the evaluated algorithms require representative computation, we excluded *rep comp* to clearly compare the performance of skyline query processing; ReSKY included subarray grouping times. ReSKY outperformed AGiDS and SKY-MR using *MBR(SG)*-based distributed processing. ReSKY terminated even faster than a single shuffling, which means that network overhead was high, and data-partitioning based distributed processing was inappropriate for the coordinator-worker model. Figure 11 shows that, as the number of subarrays increased, (1) ReSKY was more scalable than the other methods, (2) VMPSP was more efficient than OSPS because median pivots could partition the subarrays in a more balanced manner, (3) OSPS occasionally showed an unexpected performance from random pivot selection, and (4) SKY-MR-M became efficient as the

(a) MOD09A1 (b) MOD13Q1
Figure 13: Execution times according to sampling rates. (100M subarrays, 8 workers)

Table 1: The amount of data that the coordinator received in Figure 11. (MB)

	MOD09A1					MOD13Q1				
	5M	10M	15M	20M	25M	5M	10M	15M	20M	25M
ReSKY	29.4	59	64.1	81.7	96.8	227	346	494	602	672
AGiDS+(OSPS/VMPSP)	77.6	104	164	206	234	267	468	646	823	923
Shuffle	468	910	1378	1820	2288	499.4	992	1476	1962	2455
SKY-MR-M+VMPSP	661	1173	1702	2267	2989	1848	2996	3942	5281	6531
SKY-MR-S+VMPSP	698	1251	1897	2452	3054	908	1735	2564	3419	4149

Table 2: The amount of data that the coordinator received in Figure 12. (MB)

	MOD09A1					MOD13Q1				
	2	4	6	8	10	2	4	6	8	10
ReSKY	0.05	0.08	0.33	3.19	98.1	0.05	2.6	5.97	70.5	670
AGiDS+(OSPS/VMPSP)	0.002	0.04	0.41	5.82	234	0.003	5.14	11.9	160	923
Shuffle	880	1232	1584	1936	2288	951	1331	1711	2092	2455
SKY-MR-M+VMPSP	243	717	1581	2119	2957	73.1	1692	2098	2750	6719
SKY-MR-S+VMPSP	201	640	1554	2163	3051	34.1	1687	2111	2681	4138

Figure 14: Execution times according to the number of nodes. (approximately 300M subarrays, 10 attributes)

number of subarrays increased by deciding the global skyline in parallel. In Figure 12, when the number of attributes was relatively small, the Subarray-Skyline ratio was also low, and most methods terminated fast. However, the figure shows that ReSKY was more scalable as the number of attributes increased.

Figure 13 shows the execution times of ReSKY according to the sampling rates. The times did not include the S1 times to investigate the effects of the sampling clearly. We experimented with four different strategies based on Section 6.2: (1) **ReSKY-OFF** is a naïve distributed processing of ReSKY ($S1+S2+S4+S6+S7$), (2) **ReSKY-S3** is ReSKY-OFF with S3, (3) **ReSKY-S5** is ReSKY-OFF with S5, and (4) **ReSKY-S3S5** is ReSKY-OFF with both S3 and S5. We observed that ReSKY-S3 terminated earlier than ReSKY-OFF. By using global sample SG-Skyline, each worker could efficiently prune its own SGs by considering SGs in other workers with relatively small network overhead. ReSKY-S5 also terminated earlier than ReSKY-OFF

Table 3: Dataset information, SA objective score in Section 6.2, and network usage (MB) in Figure 14.

workers	2		4		6		8	
	OFF	S3S5	OFF	S3S5	OFF	S3S5	OFF	S3S5
MOD09A1								
Subarray-Skyline ratio: 1.4%, the total number of initial SGs: 3,084,534, 37.9 GB in SciDB								
Coordinator's received data	326	242	699	435	911	513	1126	557
SA objective score	1.35	1.7	1.93	3.1	2.38	4.37	2.7	5.85
MOD13Q1								
Subarray-Skyline ratio: 6.2%, the total number of initial SGs: 3,195,314, 32.6 GB in SciDB								
Coordinator's received data	1474	1207	3059	2144	4156	2549	4922	2704
SA objective score	1.38	1.67	1.94	2.97	2.42	4.27	2.85	5.68

Figure 15: Execution times according to the number of attributes in the real estate and restaurant dataset (1 worker). **Figure 16: Execution times according to the number of attributes in the real estate and restaurant dataset (8 workers).**

when the sampling rate was low; however, ReSKY-S5 tended to be slower as the sampling rate increased, the primary reason for which is larger network overhead incurred when the coordinator node gathered the sample SGs with their subarrays, despite the ability to prune more subarrays in local Subarray-Skyline. In contrast to S5, S3 required only *MBR(SG)*s without subarrays; therefore, network overhead was much smaller in S3 than in S5. Overall, ReSKY-S3S5 was the best performer. We also experimented our tuning method for ReSKY-S3S5. Specifically, each node randomly selected one chunkset and tested the different sampling rates. The tuning took average 29.34 seconds. The result shows that the tuning method can be utilized as a useful guideline.

Figure 14 shows the execution times of ReSKY according to the number of workers, $|workers|$. The S1 times included the subarray grouping times, which we separately recorded. To clearly investigate ReSKY, we recorded the net times that excluded the S1 times from the total times. ReSKY-S3S5 scaled well as $|workers|$ increased. However, the net times of ReSKY-OFF did not always decrease as $|workers|$ increased. ReSKY-S3S5 tended to become

more efficient than ReSKY-OFF as $|workers|$ increased; ReSKY-S3S5 could decrease the size of local Subarray-Skyline because global sample SG-Skyline and global sample Subarray-Skyline enabled each worker to consider data in other workers with relatively less overhead, which resulted in time reductions of S6 and S7. The graphs indicates that *rep comp* decreased considerably and easily as the $|workers|$ increased, which is one of the reasons why we proposed distributed processing. Meanwhile, subarray groups were quickly formed at query runtime, which enabled MBR-based skyline query processing.

In order to show the robustness of ReSKY, we experimented with the real estate and restaurant dataset in centralized and distributed environments, as shown in Figures 15 and 16. We excluded SKY-SB and ZSearch because of their inefficiency. Note that the results are shown on a log scale. The graphs show that ReSKY outperformed the existing methods. *rep comp* also decreased considerably in the distributed setting.

8 RELATED WORK

Array Databases. Array management techniques have been steadily researched. RasDaMan [3] has provided management and analytics for arrays from sensors, images, simulations, and statistics. SciDB [33] is a distributed array database that provides array-friendly queries and supports user-defined operators. TileDB [29] is an open-source cloud-optimized array storage engine that provides extensive APIs and tool integration. ChronosDB [40] and MARS [19] suggested array management techniques in various aspects.

Data Exploration in Array Databases. Searchlight [18] provides efficient subarray filtering by combining constraint programming solvers and main-memory synopses. PPTS [6] supports a top- k subarray query in array databases. A subarray skyline query is a novel data exploration technique that can provide insight into large-scale array data.

Skyline Query Processing. Finding Skyline was first studied in [22], and Borzsony et al. [4] introduced the skyline query in database systems. The skyline query processing methods can be divided into preprocessing-based methods [21, 24, 28, 34, 44] and preprocessing-independent methods [2, 4, 8, 13, 23, 45, 47]. We proposed the novel methods to efficiently find the relationships among *MBRs* by utilizing pivots.

Distributed Skyline Query Processing. Distributed skyline query processing for the coordinator-worker model was proposed in [31, 42, 48]. AGiDS [31] is a two-round method utilizing the dominance property among grids. Tradeoff [42] and FDS [48] focused on reducing communication costs through more rounds. Data-Partitioning based distributed skyline processing has also been studied in [16, 20, 35, 38, 39]. Distributed skyline query processing in MapReduce has been studied in [5, 27, 30, 35, 41, 43]. Our proposed methods are different from the previous studies because load balancing is achieved through *MBR*-based sampling while decreasing network overhead.

Skyline Variants. A group-based skyline was proposed in [17, 26, 46]. A group in [17, 26, 46] has a fixed number of objects; however, a subarray in our study can have a different number of objects from the other same sized-subarrays. By contrast, a subarray includes only objects within specific dimension ranges.

Region Search. Different types of region exploration have been studied in various contexts: (1) a top- k region query [6, 7, 10, 32], (2) a similar region search [11], (3) region filtering [18], (4) outlier region detection [25], and (5) a top- k region query in streaming environments [1, 12]. We proposed a subarray skyline query, which is a novel type of region exploration method.

9 CONCLUSION AND FUTURE WORKS

In this study, we proposed a subarray skyline query with efficient and scalable processing methods. We conducted the extensive experiments using an array database and real datasets and showed that ReSKY outperformed the existing methods. Possible future studies are as follows. After processing a subarray skyline query, if there are changes of cells, we can incrementally manage the query results without computing from the beginning. Acceleration through GPU is also a future work.

ACKNOWLEDGMENTS

We deeply thank the anonymous reviewers, Hyukki Lee, Jong Seon Kim, and Kijung Jung for reviewing the paper. This work was supported by (1) the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2020R1A2C2013286), (2) the MSIT under the ICT Creative Consilience program (IITP-2021-2020-0-01819) supervised by IITP, and (3) NRF grant funded by MSIT (No. NRF-2019H1D8A2105513).

REFERENCES

- [1] Daichi Amagata and Takahiro Hara. 2016. Monitoring MaxRS in Spatial Data Streams. In *EDBT 2016*. 317–328.
- [2] Ilaria Bartolini, Paolo Ciaccia, and Marco Patella. 2006. SaLSa: Computing the Skyline without Scanning the Whole Sky. In *CIKM 2006*. ACM, 405–414.
- [3] Peter Baumann, Andreas Dehmel, Paula Furtado, Roland Ritsch, and Norbert Widmann. 1998. The Multidimensional Database System RasDaMan. In *SIGMOD 1998*. ACM, 575–577.
- [4] Stephan Borzsony, Donald Kossmann, and Konrad Stocker. 2001. The skyline operator. In *ICDE 2001*. IEEE, 421–430.
- [5] Liang Chen, Kai Hwang, and Jian Wu. 2012. MapReduce Skyline Query Processing with a New Angular Partitioning Approach. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*. 2262–2270.
- [6] Dalsu Choi, Chang-Sup Park, and Yon Dohn Chung. 2019. Progressive Top-k Subarray Query Processing in Array Databases. *PVLDB* 12, 9 (2019), 989–1001.
- [7] Dong-Wan Choi, Chin-Wan Chung, and Yufei Tao. 2012. A Scalable Algorithm for Maximizing Range Sum in Spatial Databases. *PVLDB* 5, 11 (2012), 1088–1099.
- [8] Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang. 2003. Skyline with Presorting. In *ICDE 2003*, Vol. 3. 717–719.
- [9] K. Didan. 2015. *MOD13Q1 MODIS/Terra Vegetation Indices 16-Day L3 Global 250m SIN Grid V006 distributed by NASA EOSDIS Land Processes DAAC*.
- [10] Kaiyu Feng, Gao Cong, Sourav S. Bhowmick, Wen-Chih Peng, and Chunyan Miao. 2016. Towards Best Region Search for Data Exploration. In *SIGMOD 2016*. ACM, 1055–1070.
- [11] Kaiyu Feng, Gao Cong, Christian S. Jensen, and Tao Guo. 2019. Finding Attribute-Aware Similar Regions for Data Analysis. *VLDB '19* 12, 11 (2019), 1414–1426.
- [12] Kaiyu Feng, Tao Guo, Gao Cong, Sourav S. Bhowmick, and Shuai Ma. 2019. SURGE: Continuous Detection of Bursty Regions Over a Stream of Spatial Objects. *IEEE TKDE* (2019).
- [13] Parke Godfrey, Ryan Shipley, and Jarek Gryz. 2005. Maximal Vector Computation in Large Data Sets. In *VLDB 2005*. 229–240.
- [14] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. 1997. Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-Tab, and Sub Totals. *Data mining and knowledge discovery* 1, 1 (1997), 29–53.
- [15] Antonin Guttman. 1984. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD 1984*. ACM, 47–57.
- [16] Jin Huang, Feng Zhao, Jian Chen, Jian Pei, and Jian Yin. 2010. Towards Progressive and Load Balancing Distributed Computation: A Case Study on Skyline Analysis. *Journal of Computer Science and Technology* 25, 3 (2010), 431–443.
- [17] Hyeonseung Im and Sungwoo Park. 2012. Group skyline computation. *Information Sciences* 188 (2012), 151–169.
- [18] Alexander Kalinin, Ugur Çetintemel, and Stanley B. Zdonik. 2015. Searchlight: Enabling Integrated Search and Exploration over Large Multidimensional Data. *VLDB '15* 8, 10 (2015), 1094–1105.
- [19] Minsoo Kim, Ilhyun Suh, and Yon Dohn Chung. 2020. MARS: A multi-level array representation for simulation data. *Future Generation Computer Systems* 111 (2020), 419–434.
- [20] Henning Köhler, Jing Yang, and Xiaofang Zhou. 2011. Efficient Parallel Skyline Processing Using Hyperplane Projections. In *SIGMOD 2011*. ACM, 85–96.
- [21] Donald Kossmann, Frank Ramsak, and Steffen Rost. 2002. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *VLDB 2002*. VLDB Endowment, 275–286.
- [22] Hsiang-Tsung Kung, Fabrizio Luccio, and Franco P. Preparata. 1975. On Finding the Maxima of a Set of Vectors. *J. ACM* 22, 4 (1975), 469–476.
- [23] Jongwuk Lee and Seung-won Hwang. 2010. BSkyTree: Scalable Skyline Computation Using a Balanced Pivot Selection. In *EDBT 2010*, Vol. 426. ACM, 195–206.
- [24] Ken CK Lee, Baihua Zheng, Huajing Li, and Wang-Chien Lee. 2007. Approaching the skyline in Z order. In *VLDB 2007*. VLDB Endowment, 279–290.
- [25] Ki Yong Lee and Young-Kyoon Suh. 2019. A Pattern-based Outlier Region Detection Method for Two-Dimensional Arrays. *The Journal of Supercomputing* 75, 1 (2019), 170–188.
- [26] Jinfei Liu, Li Xiong, Jian Pei, Jun Luo, and Haoyu Zhang. 2015. Finding Pareto Optimal Groups: Group-Based Skyline. *PVLDB* 8, 13 (2015), 2086–2097.
- [27] Kasper Møllegaard, Jens Laurits Pedersen, Hua Lu, and Yongluan Zhou. 2014. Efficient Skyline Computation in MapReduce. In *EDBT 2014*. 37–48.
- [28] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2003. An Optimal and Progressive Algorithm for Skyline Queries. In *SIGMOD 2003*. 467–478.
- [29] Stavros Papadopoulos, Kushal Datta, Samuel Madden, and Timothy Mattson. 2016. The TileDB Array Data Storage Manager. *PVLDB* 10, 4 (Nov. 2016), 349–360.
- [30] Yoonjae Park, Jun-Ki Min, and Kyuseok Shim. 2013. Parallel Computation of Skyline and Reverse Skyline Queries Using MapReduce. *PVLDB* 6, 14 (2013), 2002–2013.
- [31] Doukeridis C. Nørveg K Rocha-Junior J.B., Vlachou A. 2009. AGiDS: A Grid-Based Strategy for Distributed Skyline Query Processing. In *Globe 2009*, Vol. 5697. Springer, 12–23.
- [32] Dimitrios Skoutas, Dimitris Sacharidis, and Kostas Patroumpas. 2018. Efficient Progressive and Diversified Top-k Best Region Search. In *SIGSPATIAL 2018*. ACM, 299–308.
- [33] Michael Stonebraker, Paul Brown, Alex Poliakov, and Suchi Raman. 2011. The Architecture of SciDB. In *SSDBM 2011*. 1–16.
- [34] Kian-Lee Tan, Pin-Kwang Eng, Beng Chin Ooi, et al. 2001. Efficient Progressive Skyline Computation. In *VLDB 2001*. 301–310.
- [35] Mingjie Tang, Yongyang Yu, Walid G. Aref, Qutaibah M. Malluhi, and Mourad Ouzzani. 2018. Efficient Parallel Skyline Query Processing for High-Dimensional Data. *IEEE TKDE* 30, 10 (2018), 1838–1851.
- [36] E. Vermote. 2015. *MOD09A1 MODIS/Terra Surface Reflectance 8-Day L3 Global 500m SIN Grid V006 distributed by NASA EOSDIS Land Processes DAAC*.
- [37] Jeffrey Scott Vitter. 1985. Random Sampling with a Reservoir. *ACM Trans. Math. Software* 11, 1 (1985), 37–57. <https://doi.org/10.1145/3147.3165>
- [38] Akrivi Vlachou, Christos Doukeridis, and Yannis Kotidis. 2008. Angle-based Space Partitioning for Efficient Parallel Skyline Computation. In *SIGMOD 2008*. ACM, 227–238.
- [39] Ping Wu, Caijie Zhang, Ying Feng, Ben Y. Zhao, Divyakant Agrawal, and Amr El Abbadi. 2006. Parallelizing Skyline Queries for Scalable Distribution. In *EDBT 2006*, Vol. 3896. 112–130.
- [40] Ramon Antonio Rodrigues Zalipynis. 2018. ChronosDB: Distributed, File Based, Geospatial Array DBMS. *VLDB '18* 11, 10 (2018), 1247–1261.
- [41] Boliang Zhang, Shuigeng Zhou, and Jihong Guan. 2011. Adapting Skyline Computation to the MapReduce Framework: Algorithms and Experiments. In *DASFAA 2011*. 403–414.
- [42] Haoyu Zhang and Qin Zhang. 2017. Communication-Efficient Distributed Skyline Computation. In *CIKM 2017*. 437–446.
- [43] Ji Zhang, Xunfei Jiang, Wei-Shinn Ku, and Xiao Qin. 2016. Efficient Parallel Skyline Evaluation Using MapReduce. *IEEE Transactions on Parallel and Distributed Systems* 27, 7 (2016), 1996–2009.
- [44] Ji Zhang, Wenlu Wang, Xunfei Jiang, Wei-Shinn Ku, and Hua Lu. 2019. An MBR-Oriented Approach for Efficient Skyline Query Processing. In *ICDE 2019*. IEEE, 806–817.
- [45] Kaiqi Zhang, Donghua Yang, Hong Gao, Jianzhong Li, Hongzhi Wang, and Zhipeng Cai. 2016. VMSP: Efficient skyline computation using VMP-based space partitioning. In *DASFAA 2016*. 179–193.
- [46] Nan Zhang, Chengkai Li, Naeemul Hassan, Sundaresan Rajasekaran, and Gautam Das. 2014. On Skyline Groups. *IEEE TKDE* 26, 4 (2014), 942–956.
- [47] Shiming Zhang, Nikos Mamoulis, and David W. Cheung. 2009. Scalable Skyline Computation Using Object-based Space Partitioning. In *SIGMOD 2009*. ACM, 483–494.
- [48] Lin Zhu, Yufei Tao, and Shuigeng Zhou. 2008. Distributed Skyline Retrieval with Low Bandwidth Consumption. *IEEE TKDE* 21, 3 (2008), 384–400.