MISE: An Array-Based Integrated System for Atmospheric Scanning LiDAR

Kyoseung Koo*
koo@dbs.snu.ac.kr
Seoul National University
Seoul, Korea

Juhun Kim*
johnjhkim@dbs.snu.ac.kr
Seoul National University
Seoul, Korea

Bongki Moon
bkmoon@snu.ac.kr
Seoul National University
Seoul, Korea

ABSTRACT

Researchers suffer from two problems while building a data processing pipeline for atmospheric scanning LiDAR. First, they must build an entire system that handles collecting signals, processing data, and visualizing the results. Second, they should support fast data processing to expand and deploy their system. In this paper, we introduce *MISE*, a fast integrated system that handles atmospheric scanning LiDAR data. *MISE* provides end-to-end processing, configuration options, and predefined signal-processing methods. In addition, the system uses an efficient chunking approach for fast processing with an array database. We demonstrate the construction and operation of a fine-dust particle monitoring system (based on a real-world scenario) using *MISE*. This demonstration demonstrates the usability and fast performance of *MISE*.

CCS CONCEPTS

• **Information systems** → **Information systems applications**; **Database design and models**.

KEYWORDS

LiDAR, Scientific Data, SciDB, Usability, Data Management

ACM Reference Format:
Kyoseung Koo, Juhun Kim, and Bongki Moon. 2021. MISE: An Array-Based Integrated System for Atmospheric Scanning LiDAR. In *33rd International Conference on Scientific and Statistical Database Management (SSDBM 2021), July 6–7, 2021, Tampa, FL, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3468791.3468829

1 INTRODUCTION

Atmospheric Light Detection and Ranging (LiDAR) is a LiDAR class used to study atmospheric properties. It is used to observe the properties of a wide area by combining it with a scanning mechanism. Wind movement measurements on the sea [8] and air pollution detection over a city [10] are examples of using atmospheric LiDAR. To perform such experiments, researchers need to construct a pipeline by fetching the signal data of the hardware, processing the data, and visualizing the atmospheric properties on their computer screen. They used a geographic information system (GIS) or built customized software to implement the data processing pipeline.

There are two problems with previous settings. First, the previous solutions are not integrated systems. Researchers need to put in additional efforts to complete pipelines. When using a GIS, they must move data from one or more hardware and save it to a database. In addition, they should build a program to process signal data to atmospheric values. If researchers choose to make a customized data processing program, they should implement many things, including preprocessing methods, utilities, and a visualization tool. Second, improving the processing speed is difficult owing to the absence of appropriate database technology. Research covering a broad area by constructing a LiDAR network [6] implies that a system is required to handle fast and big data generation. However, a relational database system (RDBMS), which is widely used to manage LiDAR data, cannot deal with this situation. Efficient LiDAR data processing using an appropriate data model is required. Studies that compensate for the first problem [4, 9] and the second one [5, 7] exist, but these cannot address the problems exactly.

In this demonstration, we introduce *MISE*, an array-based integrated system processing atmospheric scanning LiDAR data. This provides the following key features:

- End-to-end data processing: *MISE* handles data processing pipelines without installing other systems.
- Easy to update: The system provides flexible configuration options. It also provides predefined methods commonly used in signal processing so that a researcher can easily update its algorithm.
- Fast data processing: A researcher can access the processing results quickly with the benefit of an array database.

We demonstrate the operation of a fine-dust particle monitoring system using *MISE* in a real-world scenario. This includes configuring the system, automatic data processing, and visualization of the results. Through the demonstration, we show that *MISE* is convenient to use and outperforms other systems made for comparison.

2 *MISE*

To minimize a researcher's effort to build a data analysis pipeline, we provide *MISE* as *an* integrated system. In addition, *MISE* uses SciDB to support fast data processing. The integrated components and their usability are discussed in Section 2.1. We show the chunking approach in Section 2.2. Furthermore, we explain our empirical implementation issues in Section 2.3.

2.1 Integrated System

To measure atmospheric properties, researchers must handle a data-processing pipeline that comprises three steps. First, the researchers

*Both authors contributed equally to this research.

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

ACM ISBN 978-1-4503-8413-1/21/07.
https://doi.org/10.1145/3468791.3468829

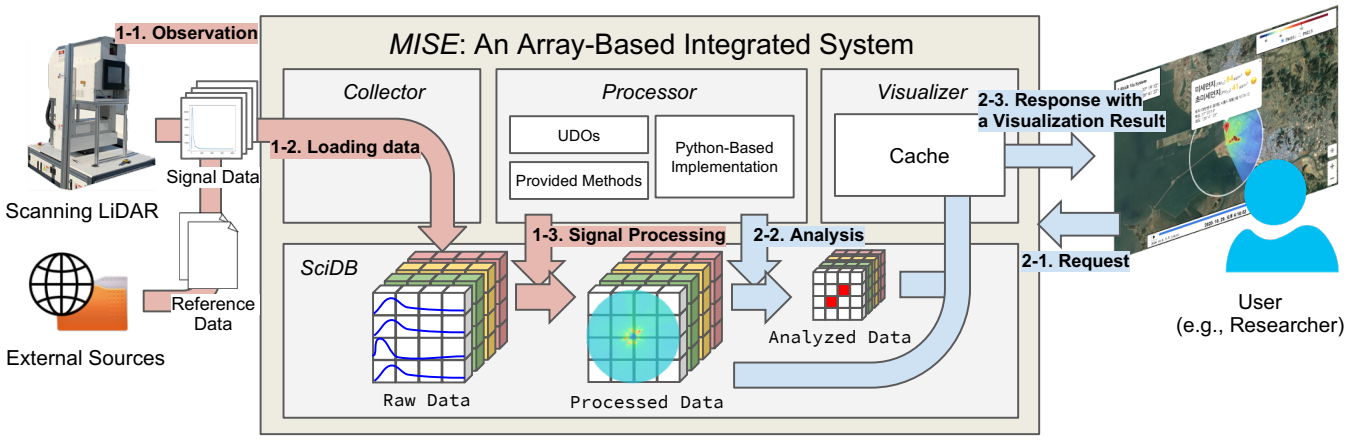

Figure 1: Overview of *MISE*. *MISE* consists of *Collector*, *Processor*, *Visualizer*, and SciDB. The entities on the left side of *MISE* are sources providing data to *MISE*. The red-colored arrows indicate the flow of data processing from the sources. A user on the right side is who wants to explore the data (e.g., a researcher). The blue-colored ones also indicate the flow of data processing.

copied the signal data from one or more LiDAR hardware to their workstations. Fetching of additional reference data is also required. Second, they wrote a program that processed raw data. Finally, they visualized the processed data using GIS or an written program.

Without using any system, too much manual work is required. If they try a new measurement, they must redo the first step. When they want to update their algorithm or visualization program, they must update their program and build them again. Even if they use GIS or previous solutions, they are still required to put in additional efforts to build the pipeline. For example, they should make loading and visualization components when using the earth observation system [9] and write a whole processing code from scratch to process signal data when using a web-based visualization platform [4].

We designed *MISE* to avoid these cumbersome works and make measurements possible after configuring it. Details of the system components are described in Section 2.1.1. In addition, Section 2.1.2 shows user experiences and *MISE*-providing features that a researcher can use when they change their measurement.

2.1.1 System Components. *MISE* comprises three modules and SciDB instances. Each module handles the corresponding pipeline step. The modules receive requests or interact with the SciDB to handle the data. Figure 1 shows an overview of *MISE*. The details of each module are as follows.

Collector loads the signal and reference data to SciDB. It watches a directory connected to a LiDAR through the network or waits for a file transmission request made from the hardware. When the data accumulated enough, the *collector* imports the data to SciDB. It also obtains reference data (e.g., the concentration of particulate matter) from external sources (e.g., sensors and open database), if needed. After loading all data, it triggers the *processor* to process raw data.

Processor processes and analyzes the signal data. The *processor* simply submits queries to SciDB. Depending on what kind of request is received, it submits processing or analysis queries. As a result, the *processor* generates a new array.

Visualizer visualizes the processed and analyzed data to the user. Once it receives a request from a user, the *visualizer* fetches the processed array from SciDB. When the user requests an analysis result, the *visualizer* triggers the *processor* to obtain the result. the *visualizer* finally returns the arrays when they are ready to serve. If the user requests values of a specific time or coordinate, the *visualizer* outputs a version for the time or values for the corresponding dimensions. the *visualizer* also caches the results to compensate for the fetching time. The built-in visualization tool provided by us is shown in Figure 2.

SciDB is used to manage and process all data in *MISE*. All data are stored in an array format in the database. To consider the time information of the data, we manage arrays with versioning.

2.1.2 Usability. *MISE* was designed for easy of use. To start *MISE*, a researcher is required to configure the options for their measurement. After running, the system automatically handles the pipeline. Once LiDAR starts a measurement and generates files, *MISE* automatically ingests, processes, and manages data. The researcher can see the processing result by accessing the *visualizer* (such as visiting the website in Figure 2).

MISE provides a Javascript Object Notation (JSON) configuration file with over 50 options including preprocessing parameters, debugging, reference data sources, etc. Users can customize the modules by modifying the file. By doing so, the user does not need to rebuild the system in such a case of changing trivial parameters or considering a geometric environment. We are currently adding more options to support more usability.

Although *MISE* provides several configuration options, researchers may need to update their processing algorithms. They may add a new preprocessing step, change the order of processing methods, or even implement a new processing algorithm. *MISE* provides Python-based and SciDB-based processing backends with predefined methods. Researchers can update the *processor* by using one of them. Python-based processing backend is researcher-friendly, but it has poor performance compared to the other. Thus, the choice is

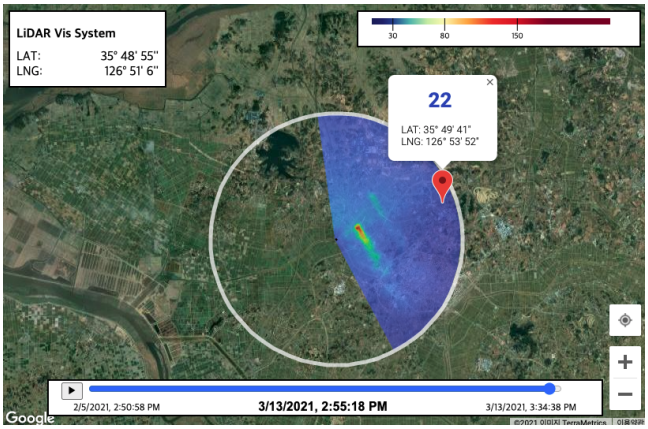

Figure 2: Built-in visualization tool. The circle in the center indicates the range of scanning LiDAR. The heat map layer inside the circle shows processed atmospheric values. If users click a specific point, the popup showing the value and coordinates is opened. On the upper side, the location of LiDAR hardware (left) and the legend (right) are displayed. At the bottom, the time travel scroll bar is presented. The play button automatically shows the results in 24 hours. Users can move and scroll in and out to explore.

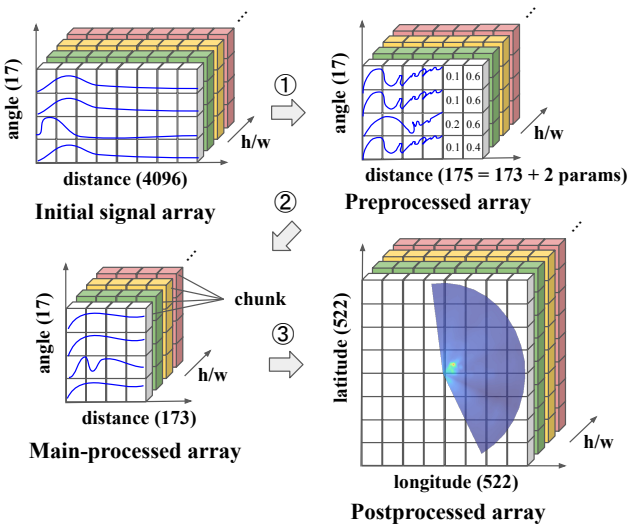

Figure 3: Example of the chunk processing. Each chunk is processed in parallel but should be processed as a batch unit. Chunks are distinguished by color. The float numbers in cells of preprocessed array are examples of the two parameters. The numbers in parentheses indicate the size of the dimension. The circled numbers show the order of three processing steps: preprocessing, main-processing, and postprocessing.

recommended when researchers develop a new algorithm or debug the system. To use Python as a processing backend, a user should change the setting to use it. Subsequently, the user can update the algorithm by modifying the Python baseline implementation provided. If the user wants to deploy the algorithm to serve, we recommend using the SciDB-based backend. The user should modify the provided SciDB UDO source code to implement an algorithm. After building them and loading the builded file to SciDB, *MISE* will use it.

We provide the following predefined methods that both backends (Python and C++) can be used:

- Interpolation: bilinear, bicubic, delaunay, and linear.
- Preprocessing: moving average, range correction, noise reduction, window summation, and integration by parts.
- Geospatial: coordinates conversion.

2.2 Chunking Approach

Improving the performance of atmospheric LiDAR data processing without considering the data characteristics is challenging. For example, a previous setting with RDBMS [5] is not appropriate for handling large scanning LiDAR datasets owing to the data's multidimensional features. Relational data models do not work well when processing geometric data with spatial and temporal dimensions [2]. As increasingly more LiDAR hardware is installed for atmospheric scanning and sending big data at once, these approaches make it difficult to achieve high performance.

MISE addresses this problem by managing scanning LiDAR data as an array model. There are several reasons why LiDAR data fit the array model. First, the raw and processed data have multidimensional features such as hardware, angle, distance, latitude, and longitude. Second, LiDAR data processing such as interpolation can benefit from utilizing locality between adjacent cells in the array. Third, we can utilize the architectural features of SciDB for performance. SciDB manages the array by each chunk, which is a part of the entire array [1]. This saves chunks distributed in several instances. When the operator runs for the array, each chunk inside the instances is processed in parallel. Therefore, we designed array schemas for efficient chunk processing by considering the LiDAR data properties.

We provide four different arrays for processing: initial signal array, preprocessed array, main-processed array, and postprocessed array. These arrays are the inputs and outputs of three processing steps: preprocessing (①), main-processing (②), and postprocessing (③). The first three arrays comprise three dimensions: hardware, angle, and distance indices. After the ③ is completed, the output postprocessed array comprises three dimensions: hardware, latitude, and longitude indices. For the chunk size, we use the entire size of each LiDAR hardware observation, the angle size multiplied by the distance size or the latitude size multiplied by the longitude size. There are no dependencies between observations of different LiDAR hardware. This implies the benefits of our chunking approach.

As an example, let us suppose an expanded situation based on real observations held in Siheung. (The number of the hardware increases from 1 to N.) Figure 3 shows the array schema and how the chunk processing works in parallel. One LiDAR hardware sends 17 directions' scanned results at once. There are 4096 distance points

for each direction. Therefore, we define the chunk size as $1 \times 17 \times 4096$ at first. After the ① and ②, the chunk size became $1 \times 17 \times 175$ and $1 \times 17 \times 173$, respectively. The reason of shrinking distance size from 4096 to 173 depends on processing algorithms. After the ③, the chunk size changes to $1 \times 522 \times 522$. By considering the distance interval, 522 is the chosen number that does not damage the visualization resolution. The parallelism between hardware is protected, even if the chunk size is changed.

2.3 Implementation Issues

We face several issues while implementing *MISE*. In this section, we explain how we solved these problems based on our experiences. First, we handle missing files transmitted from the LiDAR hardware by waiting until the end of the scanning phase. Second, we utilize caching to avoid the slow fetching time of the SciDB array. Finally, we pass the intermediate results of the previous operator to the next operator by recording them at the output array of the previous operator.

2.3.1 Handling Missing Files. LiDAR is generally installed outside the building and frequently causes file transmission failures. Retransmitting the file is one of the solutions, but it is difficult in the real-world situation for several reasons (e.g., unifying the protocols of many vendors is difficult, and the complexity of protocols increases). Thus, signal processing should be performed without missing files.

One concern is that we cannot determine whether the missing file is missing or has not arrived. In the latter case, file receiver should be waiting for a while. A simple approach is to set a timeout for each file that *MISE* should receive. However, this approach cannot be used because low latency is preferred in real-time analysis situations. Instead, *MISE* waits for all missing files for the scanning phase until the end of the phase. Data processing of scanning LiDAR can start after the end of the scanning phase. This means that the processing can wait for the not-receiving files until receiving the signal files of the last angle. One exception exists: the last files are missing. In this case, *MISE* starts processing the scanning phase after receiving at least one signal file from the next scanning phase.

2.3.2 Caching Visualization Results. *MISE* caches a rendering result of the *visualizer* as an image. Scanning and fetching a SciDB array is slow because it manages a large multidimensional array. The latency of visualizing the 2 megabytes 522×522 array is approximately 1776 ms. We solved this slow latency issue by introducing a cache holding rendered visualization results. This is reasonable because the LiDAR measurement only creates a new array (i.e. it does not affect the old arrays). After introducing a cache, we minimized the latency to approximately 120 ms.

2.3.3 Passing Parameter. When we execute two sequential operators, we often pass the output parameters of the previous one to the next one. For example, the signal-to-noise (S/N) ratio index is calculated during background noise removal, but it can be used as an input to the mass conversion equation later. A similar process is shown in Figure 3. We need to pass the output parameters of the ① to the ②. As the input and output of the UDO are arrays, the only way to pass the parameters is to utilize them. We implemented this in a naive manner by recording the parameters at the end of the distance dimension. Therefore, the distance size of the preprocessed array is 175, not 173.

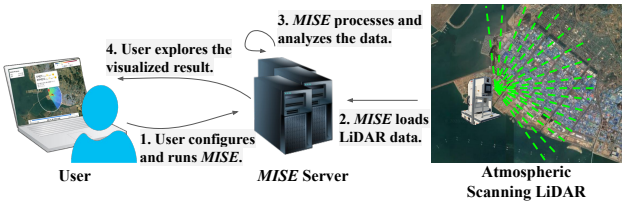

Figure 4: Demonstration Scenario

3 DEMONSTRATION

We demonstrate *MISE* in a realistic scenario. We illustrate the user interactions and automatic processes of *MISE* through this scenario. In addition, we conducted several experiments to determine that *MISE* is faster than the other systems. During the demonstration, we demonstrate the usability and performance of *MISE*.

3.1 Scenario

We built a system to analyze and visualize particulate matter levels from data collected by LiDAR in Siheung and Gimje, South Korea. Our demonstration scenario is based on this experience.

Dataset. The dataset we used is three-dimensional data comprising signal powers for the hardware, angle, and distance indices. The number of angles and distances were 17 and 4096, respectively. We set the hardware size to one to simplify the scenario. The dataset is obtained from real observations held in Siheung.

Figure 4 shows how the user utilizes *MISE*. First, the user changes the configuration file to set parameters such as the number of angles, location of the LiDAR hardware, observation range distance, atmospheric algorithm's parameters, among others. Subsequently, the user runs *MISE* to be ready for real-time analysis. Second, scanning LiDAR hardware starts observation, and data are transmitted to *MISE*. Third, when one scan of the area is completed and a certain amount of data accumulated, *MISE* starts the processing, and results are stored in SciDB. Fourth, *MISE* provides visualization results as a user requests them.

3.2 Experimental Results

We tested the performance of *MISE* compared to the Python baseline (with Numpy) and the analytic queries of the RDBMS.

Environment. The environment comprised three workstations that installed Ubuntu 16.04.6 LTS. Each workstation had an i7-4790S CPU with 8GB of memory and a 128GB SSD. They were connected through a gigabit network. We used SciDB 19.11, with four instances on each workstation. The Python and Numpy versions were 3.5 and 1.11, respectively. The version of PostgreSQL was 13.2. For a multi-node environment, we used postgres_fdw to shard the tables. The dataset was the same as that used in Section 3.1, except for the number of hardware which is 4 or 1200 depending on the comparisons.

Table 1: Evaluation Results.

(a) The processing time (second) of the ① + ② + ③ with single node and four hardware dimensions.

Python	*MISE*
86.16	**19.63**

(b) The processing time (second) of the ① + ② with 1200 hardware dimensions.

# of nodes	PostgreSQL	PostgreSQL (P4)	*MISE*
1	1920.11	551.91	**3.43**
3	3344.09	1018.59	**1.17**

To show that our chunking approach is fast, we measured the performance of *MISE* by testing the third step in Figure 4. All processes comprised ①, ②, or ③, as shown in Figure 3. We evaluated the performance of the system in two ways. First, we measured the runtime of the ① + ② + ③ for *MISE* and Python (Numpy-based). Second, we compared PostgreSQL with *MISE* by testing the ① + ② in a single- and multi-node environment (we omitted ③ because no appropriate function exists for PostgreSQL). We selected Python with Numpy, the most popular combination to develop scientific operations, and RDBMS, used in most previous approaches to handle LiDAR data, for comparison systems.

For SciDB, we defined an array *raw* having the same data schema described in Section 3.1. In addition, we implemented three UDOs for data processing in Figure 3. The ① changes *raw* to apply a conversion equation. It includes a summation for every certain point, background noise removal, moving average, and range correction. The ② transforms preprocessed data to a concentration level using the Klett formula [3]. The result is the concentration levels of the 17 directions. The ③ interpolates the outputs of the ②. After applying the UDOs for data processing, the final result is saved as an array with latitude and longitude indices for the dimensions and two particulate matter concentrations (PM10 and PM2.5) for the attributes.

For PostgreSQL, we created a *raw* table comprising four columns (hardware, angle, distance indices, and value) and a *res* table comprising five columns (similar to *raw* but having two values). We wrote analytic queries to output the same results as the SciDB UDOs. To compare parallel processing of SciDB, we also evaluated another PostgreSQL setting that executes queries in parallel (P4). We split the analytic query into four queries by hardware and execute the queries simultaneously (the P4 is the best-performing one).

Table 1 shows the results of two comparisons that evaluated the runtime of *MISE* in the scenario. *MISE* is approximately four times faster than the Python baseline implementation in processing the ① + ② + ③ (see Table 1a). Although the Python baseline is as fast as native languages (because it is powered by Numpy), *MISE*'s parallel chunk processing is faster than the Python baseline. *MISE* also outperform both PostgreSQL in a multi-node environment (see Table 1b). The SciDB chunk processing is performed in parallel for each instance, and it performs well in a multi-node environment. On the contrary, PostgreSQL obtains slow results because it requires frequent scans and joins to perform analytics queries. Interestingly, the multi-node PostgreSQL is slower than the single node. The main reason is that the analysis queries were mainly executed in the node receiving the query. The database did not push down operations to the other nodes, so the nodes only perform scans and insertion.

4 CONCLUSION

We introduce *MISE* that efficiently manages the atmospheric scanning LiDAR data. Our system provides end-to-end data processing, thereby helping researchers minimize their efforts while building a complete system. By using an array-based chunking approach, we benefit from processing and analyzing the data. In this demonstration, we construct and operate an *MISE*-based system for a scenario that mimics South Korean cases. *MISE* exhibits high usability and fast performance in a series of scenario steps.

ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea (2020R1A2C1010358 and 2016M3C4A7952633).

REFERENCES

[1] Paul G Brown. 2010. Overview of SciDB: large scale array storage, processing and analysis. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. 963–968.
[2] Philippe Cudre-Mauroux, Hideaki Kimura, Kian-Tat Lim, Jennie Rogers, Samuel Madden, Michael Stonebraker, Stanley B Zdonik, and Paul G Brown. 2010. SS-DB: A Standard Science DBMS Benchmark. *Extremely Large Databases Conference* (2010).
[3] James D Klett. 1981. Stable analytical inversion solution for processing lidar returns. *Applied optics* 20, 2 (1981), 211–220.
[4] Paul Lewis, Conor P Mc Elhinney, and Timothy McCarthy. 2012. Lidar data management pipeline; from spatial database population to web-application visualization. In *Proceedings of the 3rd International Conference on Computing for Geospatial Research and Applications*. 1–10.
[5] Luboš Matějíček, Pavel Engst, and Zbyněk Jaňour. 2006. A GIS-based approach to spatio-temporal analysis of environmental pollution in urban areas: A case study of Prague's environment extended by LIDAR data. *Ecological Modelling* 199, 3 (2006), 261–277.
[6] Gelsomina Pappalardo, Aldo Amodeo, Arnoud Apituley, Adolfo Comeron, Volker Freudenthaler, Holger Linné, Albert Ansmann, Jens Bösenberg, Giuseppe D'Amico, Ina Mattis, et al. 2014. EARLINET: towards an advanced sustainable European aerosol lidar network. *Atmospheric Measurement Techniques* 7, 8 (2014), 2389–2409.
[7] Gary Planthaber, Michael Stonebraker, and James Frew. 2012. EarthDB: scalable analysis of MODIS data using SciDB. In *Proceedings of the 1st ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*. 11–19.
[8] Susumu Shimada, Jay Prakash Goit, Teruo Ohsawa, Tetsuya Kogaki, and Satoshi Nakamura. 2020. Coastal Wind Measurements Using a Single Scanning LiDAR. *Remote Sensing* 12, 8 (2020). https://doi.org/10.3390/rs12081347
[9] Zhenyu Tan, Peng Yue, and Jianya Gong. 2017. An array database approach for earth observation data management and processing. *ISPRS International Journal of Geo-Information* 6, 7 (2017), 220.
[10] Jinhong Xian, Dongsong Sun, Wenjing Xu, Yuli Han, Jun Zheng, Jiancao Peng, and Shaochen Yang. 2020. Urban air pollution monitoring using scanning Lidar. *Environmental Pollution* 258 (2020), 113696.