electronics

MDPI

Article

The Distributed HTAP Architecture for Real-Time Analysis and Updating of Point Cloud Data

Juhyun Kim and Changjoo Moon *

Department of Smart Vehicle Engineering, Konkuk University, Seoul 05029, Republic of Korea; rlawngus1224@konkuk.ac.kr

* Correspondence: cjmoon@konkuk.ac.kr

Abstract: Updating the most recent set of point cloud data is critical in autonomous driving environments. However, existing systems for point cloud data management often fail to ensure real-time updates or encounter situations in which data cannot be effectively refreshed. To address these challenges, this study proposes a distributed hybrid transactional/analytical processing architecture designed for the efficient management and real-time processing of point cloud data. The proposed architecture leverages both columnar and row-based tables, enabling it to handle the substantial workloads associated with its hybrid architecture. The construction of this architecture as a distributed database cluster ensures real-time online analytical process query performance through query parallelization. A dissimilarity analysis algorithm for point cloud data, built by utilizing the capabilities of the spatial database, updates the point cloud data for the relevant area whenever the online analytical process query results indicate high dissimilarity. This research contributes to ensuring real-time hybrid transactional/analytical processing workload processing in dynamic road environments, helping autonomous vehicles generate safe, optimized routes.

Keywords: autonomous vehicles; point cloud data analysis; hybrid transactional/analytical processing (HTAP); distributed database cluster; spatial database; light detection and ranging (LiDAR)

Citation: Kim, J.; Moon, C. The Distributed HTAP Architecture for Real-Time Analysis and Updating of Point Cloud Data. *Electronics* **2023**, *12*, 3959. https://doi.org/10.3390/electronics12183959

Academic Editors: Calin Iclodean, Bogdan Ovidiu Varga and Felix Pfister

Received: 10 August 2023
Revised: 17 September 2023
Accepted: 18 September 2023
Published: 20 September 2023

1. Introduction

For autonomous vehicles, the rapid perception of changes in the surrounding environment is essential. Substantial research on the perception of surrounding environmental changes through various sensors is ongoing, in particular research using LiDAR (light detection and ranging) sensors. LiDAR sensors use laser pulses to extract information about objects in the surrounding environment as three-dimensional (3D) point cloud data. Autonomous driving technology has been developed that uses these point cloud data to cope with potential obstacles or sudden situations [1]. More broadly, autonomous driving technology using point cloud data helps autonomous vehicles create safe, optimized local routes in rapidly changing road traffic situations. While autonomous driving route creation technology using the Global Positioning System (GPS) is very useful, GPS signal is frequently interrupted or inaccurate in the real world. In such cases, an autonomous vehicle must generate its route using a 3D precision map created from point cloud data for their surroundings collected from LiDAR sensors [2]. The currency of these point cloud data is a crucial factor in creating safe routes for autonomous vehicles. Figure 1 illustrates an example of subsequent vehicles altering their paths through a point cloud map server that ensures point cloud data currency. Figure 1 represents the following process: (1) When a preceding vehicle navigates based on the existing map and detects changes, (2) the server updates the point cloud map to incorporate these changes. (3) After distributing the updated map, (4) subsequent vehicles adjust their paths, enabling safe autonomous driving.

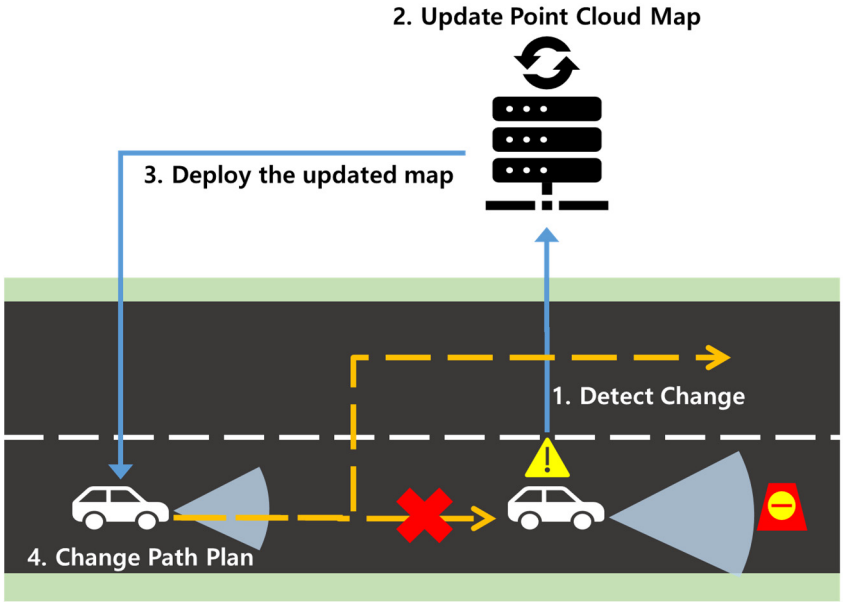

Figure 1. An example of safe autonomous driving through point cloud map updating.

Therefore, a system is needed to manage the most up-to-date set of point cloud data. This system should not only store the latest point cloud data but also be able to fetch and read the most up-to-date data in real-time, enabling autonomous vehicles to use these data for route creation. Most of the existing research on this topic has used Hadoop as a platform to process point cloud data. However, Hadoop is characteristically incapable of modifying the loaded data, meaning that the data must instead be deleted and reloaded [3]. This incapacity presents the disadvantage of incurring significant computational costs to maintain the point cloud data in their most up-to-date state.

In response to these issues, this paper proposes a distributed HTAP architecture using a spatial database. While conventional databases are specialized for transactions, allowing for fast reading and writing of data, their storage and search efficiencies decrease as the volume of data increases, making them unsuitable for data analysis or aggregation [4]. However, the architecture proposed herein uses both columnar tables and row-based tables, facilitating real-time analysis and transactions. Row-based tables, recording data on the disk for each row, are specialized for reading and writing capabilities, while columnar tables, recording data on the disk for each column, are specialized for aggregation and columnar operations. Based on these characteristics, we construct an architecture that allows simultaneous analysis and transactions by implementing the following functions: (1) Point cloud data and GPS data collected from vehicles are transmitted to the system in real-time. (2) The system uses the vehicle's GPS coordinates to determine if there is point cloud data collected from coordinates within 5 m of the vehicle's coordinates. (3a) If no such data exist, the transmitted data are inserted and stored in the map table. (3b) If such data already exist, the dissimilarity between the existing data and the new data is computed. (4) If the dissimilarity value exceeds a certain threshold, the existing data values are updated to the transmitted data values.

The architecture proposed in this paper is HTAP (hybrid transactional/analytical processing) architecture that can simultaneously handle the OLTP (online transactional process) for point cloud data transmitted in real-time and the OLAP (online analytical process) for comparative analysis between the transmitted point cloud data and the point cloud data that generated the existing precision map. Considering the size of each set of point cloud data, performing real-time transaction processing and analysis processing on a single node is untenable. Therefore, this architecture was designed based on Citus, a distributed software extension of PostgreSQL version 15.2, and was constructed with three worker nodes and one master node.

Unlike existing systems, this system has the unique feature of using columnar storage for high-cost point cloud analysis queries, thus enabling compression and parallelization. Additionally, as the number of vehicles collecting data or the volume of collected data increases, the system can maintain real-time performance by linearly increasing the number of nodes through its data rebalancing function. In this paper, we construct this architecture and determine whether the point cloud data update function based on the point cloud data analysis results operates correctly. To evaluate the performance of the architecture, we measure the OLTP/OLAP query response time and data compression efficiency. Furthermore, we compare data compression rates between our architecture's storage type and different storage types, and we evaluate changes in OLTP/OLAP query response times resulting from an increase in the number of nodes within our architecture's configuration.

The main contributions of our study can be summarized as follows:

- We constructed a HTAP (hybrid transactional/analytical processing) architecture that inserts, analyzes and updates point cloud data in real-time using distributed database technology.
- The architecture has a feature that autonomously detects changes over time by determining the dissimilarity between sequential runs through the environment and maintaining and managing the latest state of point cloud data.
- We evaluate whether this architecture is suitable for continuous operation by measuring OLTP/OLAP query response time and data compression efficiency.

The structure of the remainder of this paper is as follows: Section 2 introduces related research and background knowledge. Section 3 describes the architecture structure and the algorithm by which the architecture operates. Section 4 explains the evaluations conducted and their results, and, finally, Section 5 concludes with an analysis of the results from Section 4.

2. Background and Related Work

2.1. HTAP (Hybrid Transactional/Analytical Processing)

HTAP, first defined by Gartner Inc. in 2014, is a data management architecture aimed at simultaneously processing transactions on the most recently received data while addressing workloads for real-time analytics [5]. HTAP was introduced to surpass traditional data management systems, whose architecture sacrificed real-time capabilities when analyzing data, due to the use of separate systems for transaction processing (OLTP) and analytical processing (OLAP) while bearing the costs of ETL (extract, transform, load) operations. HTAP systems aim to combine the benefits of both OLTP and OLAP within a single system, mitigating these sacrifices.

In subsequent HTAP research, Huang et al. proposed the HTAP database TiDB, constructed through a solution that separately manages data for OLTP and data for OLAP using replication technology based on the Raft algorithm [6]. Yang et al. further proposed the HTAP database F1 Lightning, constructed through a solution that merges the change logs (delta) of tables into one consistent view [7]. These studies were significant in that they built environments in which OLTP queries and OLAP queries did not interfere with each other in their contributions to HTAP workloads. However, both studies share a key limitation: they did not consider the HTAP workload processing capability for point cloud data, including spatial data types and spatial query functions. In another study related to HTAP, Aunn Raza et al. proposed a scheduling algorithm for hybrid transaction/analytical processing (HTAP) systems, addressing runtime data freshness and performance requirements through elastic resource management [8]. They improved OLAP query performance and reduced OLTP throughput by designing an in-memory system. Additionally, Zhenghua Lyu et al. introduced the Greenplum HTAP database, which supports both OLTP and OLAP workloads while minimizing performance overhead and maintaining ACID properties [9]. They incorporated a global deadlock detector into Greenplum to enhance query processing concurrency by employing different commit protocols based on the situation. While the aforementioned studies focus on enhancing OLTP and OLAP performance in

databases, they are not specifically related to spatial databases. In contrast, Anh-Vu Vo et al. presented a study on adopting the Hadoop HBase framework for managing large-scale point cloud data [10]. They justified the use of Hadoop HBase as a distributed key-value database system suitable for managing spatiotemporal data. However, in this research, the requirement for aggregating and processing individual spatial points in real-time led to the use of distributed PostgreSQL instead of Hadoop.

2.2. Point Cloud and Spatial Database

A point cloud refers to a dataset composed of a large number of points in a 3D coordinate system, collected using LiDAR. Associated with each point are various pieces of information, including (x, y, z) coordinates and a reflectance value, making such datasets generally large in size. Therefore, such datasets typically undergo a down-sampling process using a voxel filter before processing and analysis.

A spatial database is a database that explicitly supports spatial object types and spatial query functions not explicitly supported in standard databases. A spatial database recognizes coordinate data input not as a simple string but as an object, allowing spatial query performance to be optimized.

Recent research on updating 3D precision maps constructed from point cloud data has been diverse. Mohammad et al. proposed a technique to automatically update the building map database based on automatic detection of events such as the emergence of a new building or changes in existing buildings from point cloud data collected from LiDAR sensors [11]. However, the database does not directly store and process point cloud data about buildings but only stores two-dimensional (2D) information in a top-view format, making it challenging to apply to the dynamic autonomous driving environments considered in the present study. Lee et al. proposed a fog computing server model that applied the normal distributions transform—simultaneous localization and mapping (NDT–SLAM) technique to LiDAR data generated from vehicles to create an HD (high-definition) map [12]. This study was significant, as it used Hadoop to store large amounts of point cloud data and created precision maps using SLAM techniques. However, both Hadoop and SLAM are quite high in computational cost, making real-time data updating unsustainable. Lewis et al. proposed a pipeline to manage LiDAR data using a spatial database [13]. According to his study, the appropriate method to load LiDAR data into a spatial database is the parallel technique, which reduces the time required by more than 10% compared to the direct or copy command. This research was significant as it compared the time required for various techniques, but it considered only the loading of data into the database and not the presentation of data analysis results or processing results. In another study, Lee et al. proposed an information management system for road events using a spatial database by dividing roads into links and nodes [14]. This study implemented a feature that delivered only those events occurring on a specific link to vehicles that could potentially enter that link using the functions of PostGIS, an extension of PostgreSQL. However, this system did not load point cloud data into a database, so analysis and operations related to point clouds could not be conducted in real-time. Furthermore, Juan et al. have developed a labeled database for autonomous driving systems using point cloud data captured through LiDAR [15]. He processed overlapping areas of the images, segmented each map, and built a database to enable neural networks to detect objects. However, he manages the database with two-dimensional top-view maps. Our research differs in that we directly manage and update three-dimensional map data in the database. Kang et al. conducted a study on real-time semantic segmentation of LiDAR-based point cloud data for autonomous driving scene recognition [16]. They employed a convolutional neural network (CNN) approach that projects 2D LiDAR data into 3D LiDAR data, ensuring real-time performance and enabling real-time object recognition and decision-making. However, their study differs from ours in that they do not generate and update maps using LiDAR, setting it apart from our research.

3. Distributed HTAP Architecture for Point Clouds

In this section, we provide an explanation of the configuration of the suggested framework and the algorithm utilized for data processing. In Section 3.1, we provide an overview of the overall structure of the system architecture developed for analyzing and updating the data collected from vehicles. In Section 3.2, we describe the structure of the distributed database cluster built to handle the HTAP workload. In Section 3.3, we detail the data flow, from collecting LiDAR data from vehicles to the final data update process. Lastly, in Section 3.4, we explain the algorithm for comparing and analyzing point cloud data based on the dissimilarity between the existing dataset and the new dataset.

3.1. System Architecture

The proposed system aims to update the point cloud map in real-time using data collected from data collection vehicles. Accordingly, the system architecture of this study is divided into the Edge part, which collects LiDAR and GPS data, and the Cloud part, which is a data platform that analyzes and updates the data, as shown in Figure 2.

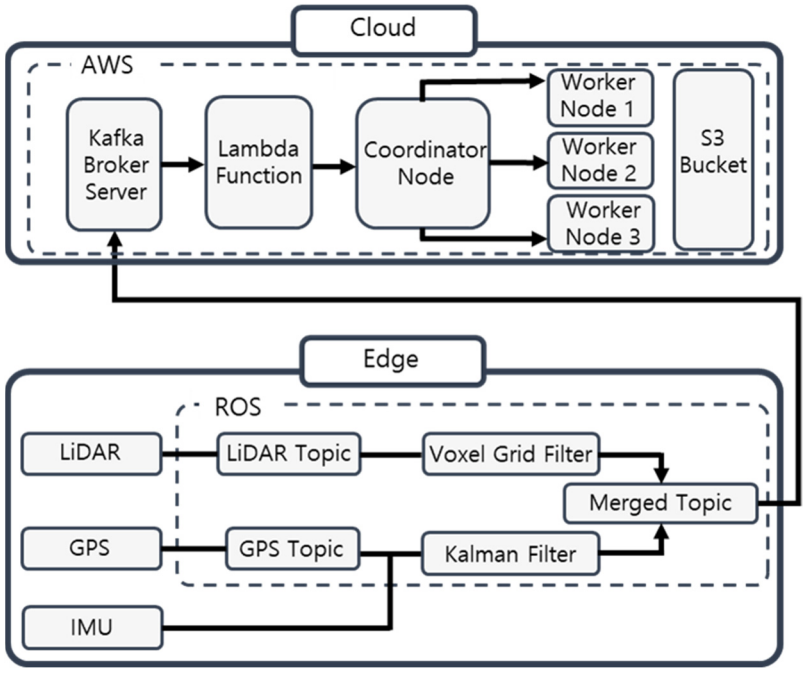

Figure 2. System architecture.

3.1.1. Edge Part

For the Edge part, we established a sensor data collection environment using the ROS (Robot Operating System) middleware [17], which provides a framework for networking robotic sensor components. Then, preprocessing was conducted on the collected data. The LiDAR topic, which represents the surrounding environment and was collected from the vehicle's LiDAR sensor, was down-sampled using the voxel grid filter. Points that captured the shape of the observed vehicle were filtered out using masking to ensure that they were excluded from the data [18]. The topic detailing the vehicle's position and direction, gathered from the GPS sensor, was denoised using a Kalman filter. Subsequently, these ROS topics were transmitted to the broker cluster, a system responsible for managing and distributing messages, as Kafka topics through a Kafka–ROS Connector [19], which facilitates seamless integration between the ROS and Kafka platforms.

3.1.2. Cloud Part

For the Cloud part, all systems were built using cloud computing services to enable proactive scaling. When a topic was delivered to the Kafka broker server, AWS (Amazon

Web Services) Lambda [20], a serverless function triggered based on that event, acted as a consumer to process the data and then inserted it into the distributed database cluster. The distributed database cluster consisted of a total of three nodes: one coordinator node and two worker nodes. A pipeline was set up so that the Lambda function stored the data in an S3 bucket, which is a scalable storage service offered by AWS, ensuring future access to the original point cloud data.

3.2. Distributed Database Cluster for HTAP Workloads

In this study, the distributed database cluster was constructed using the Citus distributed solution based on PostgreSQL. We chose Citus because it supports columnar tables within the given database, and its query optimizer offers distributed optimization for analytical queries. With these features, our system was designed to manage both row-based and columnar tables within a single database, enabling real-time analysis of large-scale data and immediate transaction processing. Additionally, we utilized the Point_Cloud and PostGIS extensions supported by the open-source PostgreSQL to handle point cloud data types and functions [21]. For the role of a connector, which allows the bulk transfer of external point cloud data to PostgreSQL, we employed the C++-based Point Data Abstraction Library (PDAL) [22]. Figure 3 shows the structure of the distributed database cluster capable of handling the HTAP workloads shown in Figure 2. This HTAP architecture, namely the distributed database cluster, is composed of a coordinator node and three worker nodes. The coordinator node holds metadata related to the distributed tables on each worker node. When queries are executed on distributed tables, the coordinator node, through the Query Optimizer, forwards the queries to each node and aggregates the results. This approach enhances the response time of queries that involve extensive exploration or computation. Considering the size of the data used in this study, it was determined that a cluster of three nodes would be suitable for distributing the workload effectively. Therefore, three worker nodes were established with the same configuration.

HTAP architectures typically use row-based storage as the standard for data input, with columnar storage replicating the data to handle HTAP workloads. However, the architecture presented in this paper takes a different approach. This architecture batch-inserts data into the columnar table and then, after comparing with the existing data, updates the row-based table. The reason this approach is feasible is because we significantly reduced the duration of OLAP queries by parallelizing them and arranging columns with low cardinality. By placing columns with low cardinality, the system enhances the performance of high-cost aggregate queries. Detailed insights into the data flow and algorithms are provided in Sections 3.3 and 3.4.

The point_cloud_analysis table, in which data are most frequently stored and aggregate and analysis queries are run regularly, was constructed as a distributed columnar table. The "point_cloud_analysis" table was designed with the "sector" column as the distribution key. This decision was made because the table mainly executes queries that involve aggregation and computations based on sectors. In contrast, the point_cloud_map table, which frequently undergoes upsert (update or insert) and read transaction operations, was built as a distributed row-based table. Considering that the table involves both deletion and insertion operations based on the "ref_id" column, the "ref_id" column was chosen as the distribution key. Additionally, since there are frequent searches based on the GPS column, a generalized search tree (GiST) index has been set on the GPS column. The GIST index, short for Generalized Search Tree index, is a data structure used in databases to efficiently manage spatial data and enable quick spatial queries, such as range searches and nearest neighbor searches [23]. The gps_position_log, imu_pose_log, and point_cloud_analysis_hist tables were constructed as row-based tables on the coordinator node without distributing them. This decision was made not only because these tables are relatively small but also because large-scale operations frequently use the data within these tables as a reference. If these tables were to be distributed, there would be a risk of network overhead, so these tables are composed of a single table belonging to the coordinator node.

Figure 4 shows the entity relationship diagram for these tables. All tables possess a column named ref_id. The ref_id functions as an identifier allocated to data in accordance with the vehicle's LiDAR data collection cycle, representing unique values for reported time, GPS coordinates, and heading angle. When a vehicle collects Point Cloud Data using LiDAR, the reported time, GPS coordinates, and heading angle of the vehicle are associated with a ref_id and written into the gps_position_log table. The corresponding IMU data at that moment is recorded in the imu_pose_log table, while the generated histogram is written into the point_cloud_analysis_hist table. As each of these three tables contains data rows that are uniquely associated with ref_id, they use ref_id as their primary key. Both the point_cloud_map and point_cloud_analysis tables share the same ref_id for all points captured at that moment. The point_cloud_map table consists of a point cloud data column, an updated_at column indicating when the data was last updated, and the GPS coordinates of the vehicle. Lastly, the point_cloud_analysis table is configured to categorize point cloud data based on the location of points using the sector column, enabling swift aggregation.

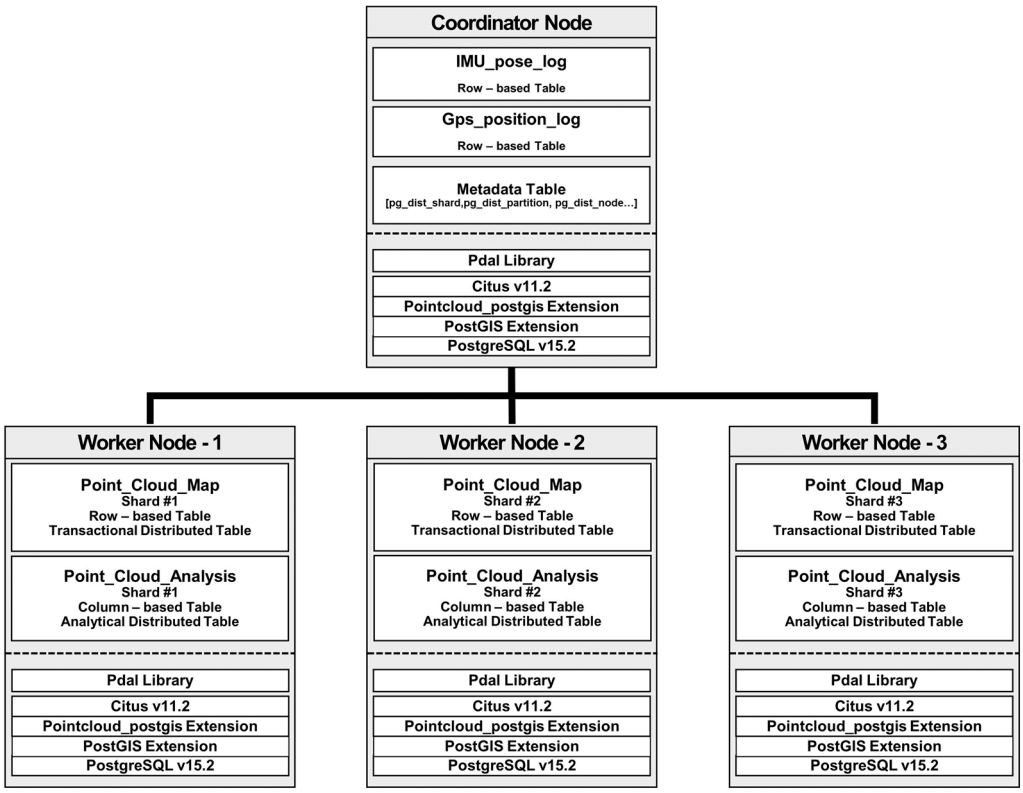

Figure 3. Distributed database cluster for hybrid transactional/analytical processing (HTAP) workloads.

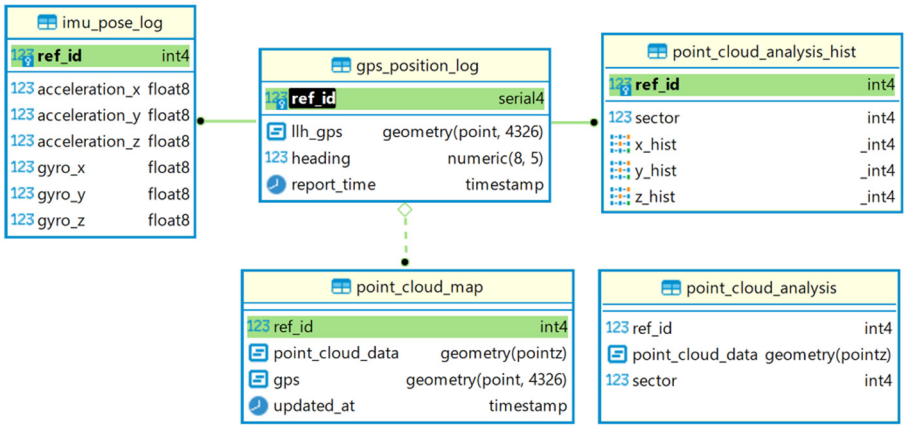

Figure 4. Entity relation diagram.

3.3. Data Flow in HTAP Architecture

The primary objective of the system constructed in this study is to analyze data collected from LiDAR, the inertial measurement unit (IMU), and GPS sensors in real-time through the HTAP architecture. Based on the analysis results, the system updates and maintains the point cloud dataset in its latest version. This subsection provides a detailed explanation of the data processing pipeline designed to achieve this goal. Figure 5 shows the process of data handling within the system presented in this study. The detailed steps of this flow chart are described below.

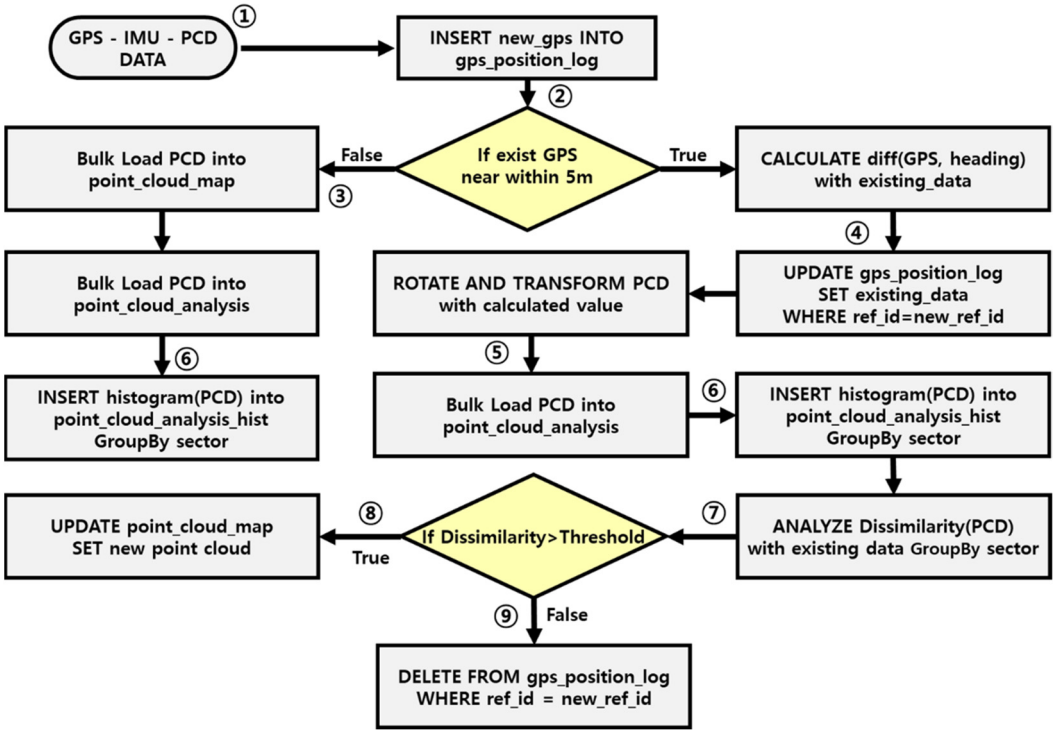

Figure 5. Data flow chart.

1. Data from LiDAR, GPS, and IMU sensors, collected from each vehicle operating in the ROS environment, are generated at 1-s intervals. These data undergo preprocessing in the Edge part of the architecture and are then forwarded to Lambda. PCD data undergo down-sampling using the voxel grid filter, and points corresponding to the observing vehicle and ground points are removed via the mask filter. GPS data are refined for accuracy and noise reduction using IMU data and a Kalman filter. The preprocessed data are sent to the Kafka broker server, from which the serverless function Lambda, acting as a Kafka consumer, receives it. Subsequently, Lambda, which has access to the Citus cluster, sends a query related to data processing to the coordinator node.
2. Lambda inserts the GPS and IMU data into the gps_position_log and imu_pose_log tables, respectively, and retrieves a new ref_id. It then searches for previously inserted vehicle coordinates within 5 m of the new ref_id's GPS data using the ST_DWithin function. If a significant quantity of data is found, Lambda retrieves the existing ref_id, GPS, and heading values of the physically closest and most recently inserted data.
3. If no such data are found, the point_cloud_map and point_cloud_analysis tables are immediately bulk-loaded with point cloud data.
4. If such data do exist, the differences are calculated between these data and the LLH coordinates and heading values from the GPS sensor. The data corresponding to the new ref_id in the gps_position_log table are then updated with the GPS and heading data corresponding to the existing ref_id.

5. Based on the GPS and heading differences calculated in step 4, Euler rotation and coordinate transformations are performed on the new point values. The angles formed by the points based on the heading are calculated to distinguish sectors. The transformed data are then bulk-loaded into the point_cloud_analysis table.
6. When bulk-loading point cloud data in step 5, a histogram of those point cloud data is generated. Distributions of points along the x-, y-, and z-axes are created and inserted into the point_cloud_analysis_hist table.
7. In the point_cloud_analysis table, the points corresponding to existing ref_id and new ref_id are aggregated by sector, and then the difference in average distance between each point and origin and the Euclidean distance between the two histograms is measured. This OLAP query is executed only for points within 30 m, and a standard threshold is set. If the dissimilarity exceeds this threshold, the comparator returns true. Otherwise, it returns false. This comparison algorithm is discussed in detail in Section 3.4.
8. If the result of step 7 is true, the data corresponding to the existing ref_id in the point_cloud_map table are updated with the data for the new ref_id, and the updated_at column records the update time.
9. If the result of step 7 is false, the system determines that no changes have occurred. Therefore, within the gps_position_log table, data corresponding to the new ref_id are removed, while data within the point_cloud_map table are preserved without updating.

3.4. Algorithm for Comparing and Analyzing Point Clouds

This subsection explores the algorithm for comparing and analyzing point cloud data, as introduced in Section 3.3.

Initially, new point cloud data captured within a 5-m radius of the existing ref_id's position undergo Euler rotation and coordinate transformation. This step ensures that every point aligns with the coordinate system of the data corresponding to the existing ref_id. Subsequently, the transformed point cloud data points are allocated to sectors, based on the angle measured clockwise in plan view from the reference point (0, 0) as the origin and true north, in increments of 30 degrees. Given that our system conducts OLAP operations on points with significantly high cardinality, we introduced a sector column with a relatively low cardinality of 12. Using this column as a reference, the query optimizer parallelizes the OLAP query, thereby reducing the computation time. Table 1 shows the query response time based on the presence or absence of the sector column for join operations. This query aggregates the number of points for both the existing ref_id and new ref_id. For each of the two distinct pairs of ref_ids presented, the response time reduction associated with the presence of the sector column was 4.836 s (for existing ref_id 1 and new ref_id 2) and 4.746 s (for existing ref_id 3 and new ref_id 4). This response time reduction corresponded to a decrease of approximately 65%.

Table 1. Online analytical process (OLAP) query response time comparison.

Use Sector Column	New ref_id	Existing ref_id	Number of New Points	Number of Existing Points	Response Time (s)
No	2	1	34,345	36,021	7.492
Yes	2	1	34,345	36,021	2.656
No	4	3	38,813	32,796	6.961
Yes	4	3	38,813	32,796	2.215

The point cloud data loaded into the point_cloud_analysis table are aggregated with points from the existing ref_id from within the same sector and within 30 m of the origin. For each sector, the average distance of points to the origin is measured and then squared. Subsequently, three histograms are generated based on the distribution of points within the same sector for the x-, y-, and z-axes, and the differences between these histograms

and the histograms representing the existing ref_id are calculated. In this system, the histogram divides each axis within a sector into 59 sections from −30 m to +30 m, summing up the number of points in each section, resulting in an array of total length 60. Figure 6 depicts a top-view perspective of the vehicle, and the histogram shown in the illustration represents the distribution based solely on the x-values of points within the same sector, disregarding y and z values. Figure 6 shows the points within sector 2 categorized into 1-m intervals along the *x*-axis, representing the count of points within each interval using a one-dimensional array. The illustration provides an example for better understanding, using an array of length 6. However, in reality, a 60-length array ranging from −30 m to +30 m is employed for representation. This approach allows for the abstraction of complex 3D point distributions into simple arrays of three 1D histograms, reducing computation costs accordingly. Through this histogram, the clustering pattern of the points can be represented as an array, making meaningful comparisons feasible.

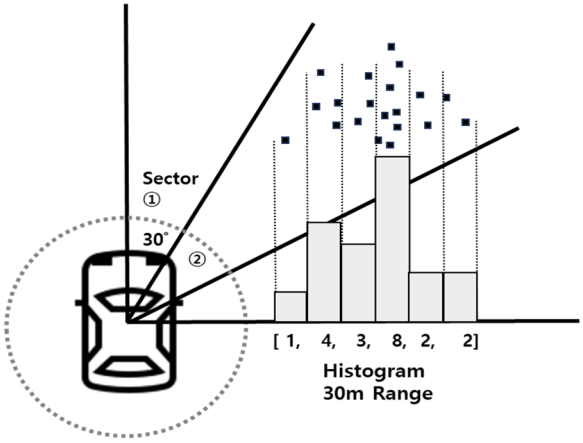

Figure 6. An example depicting the *x*-axis point cloud histogram of the sector 2 area.

The metric for assessing the dissimilarity of point cloud data in this paper is expressed as shown in Equation (1). The dissimilarity between two ref_ids is quantified using the product of two quantities: (1) the square of the difference between the existing ref_id and the new_ref_id for the average distance of the point to the origin, and (2) the sum of the Euclidean distances for the x-, y-, and z-axes histograms between the existing ref_id and the new ref_id. We formulated a metric by multiplying the histogram, which represents the point distribution pattern with the difference in the average distance representing the distance to the points. This metric is designed to detect changes unless the distribution of points is similar and the average distance is consistent.

$$\begin{aligned} \text{Dissimilarity} = {} & \left(\text{avgDistance}_{\text{existing}} - \text{avgDistance}_{\text{new}}\right)^2 \times \\ & \{\sqrt{\textstyle\sum_{i=1}^{60}\left(\text{xHistogram}_{\text{existing}}[i] - \text{xHistogram}_{\text{new}}[i]\right)^2} + \\ & \sqrt{\textstyle\sum_{i=1}^{60}\left(\text{yHistogram}_{\text{existing}}[i] - \text{yHistogram}_{\text{new}}[i]\right)^2} + \\ & \sqrt{\textstyle\sum_{i=1}^{60}\left(\text{zHistogram}_{\text{existing}}[i] - \text{zHistogram}_{\text{new}}[i]\right)^2}\} \end{aligned} \tag{1}$$

The pseudocode for the OLAP query code corresponding to the above algorithm is shown in Algorithm 1. Histogram construction was implemented using the width_bucket function, and it was managed as an array using the array_agg function.

Algorithm 1: Pseudocode of online analytical process (OLAP) query for comparing point clouds

```
WITH
    new_data AS (
        SELECT sector,
        AVG(ST_Distance(point_cloud_data, ST_MakePoint(0, 0))) AS avg_distance
        FROM point_cloud_analysis
        WHERE ref_id = β
         AND ST_Distance(point_cloud_data, ST_MakePoint(0, 0)) < 30
        GROUP BY sector
    ),
    existing_data AS (
        SELECT sector,
        AVG(ST_Distance(point_cloud_data, ST_MakePoint(0, 0))) AS avg_distance
        FROM point_cloud_analysis
        WHERE ref_id =α
         AND ST_Distance(point_cloud_data, ST_MakePoint(0, 0)) < 30
        GROUP BY sector
    ),
    euclide_distance AS (
    SELECT
            EuclideDistance(new_histogram.x, existing_histogram.x) as euc_dist_x,
            EuclideDistance(new_histogram.y, existing_histogram.y) as euc_dist_y,
            EuclideDistance(new_histogram.z, existing_histogram.z) as euc_dist_z
        FROM point_cloud_analysis_histogram AS new_histogram
        JOIN point_cloud_analysis_histogram AS existing_histogram
         ON new_histogram.ref_id = β
        WHERE existing_histogram.ref_id = α
    )
SELECT
    CASE
        WHEN EXISTS (
            SELECT 1
            FROM new_data
            JOIN existing_data ON new_data.sector = existing_data.sector
            CROSS JOIN euclide_distance
            WHERE Power(new_avg_distance − existing_avg_distance) *
            (euc_dist_x + euc_dist_y + euc_dist_z) > THRESHOLD
        ) THEN TRUE
        ELSE FALSE
    END AS change_detected
```

The OLAP query code used consists of three CTEs (common table expressions). The CTEs that calculate the average distance of aggregated points by sector from the point_cloud_analysis table are named new_data and existing_data. Additionally, a CTE named euclid_distance is designated to retrieve the histogram corresponding to the two ref_ids from the point_cloud_analysis_hist table and calculate the Euclidean distance between the two histograms. Subsequently, using the received new_ref_id and existing_ref_id, the average distance difference by sector and the Euclidean distance are computed. These computed values are then used in the form of the previously described Equation (1) to calculate the dissimilarity. If the dissimilarity exceeds the threshold in any case, the data corresponding to the existing ref_id in the point_cloud_map table are updated with the new data.

4. Evaluation of HTAP Architecture

In this section, we conduct qualitative and quantitative evaluations of the proposed architecture. In Section 4.1, we outline the scenarios for both qualitative and quantitative evaluations. Section 4.2 presents the results of the qualitative evaluation obtained from

our experiments, with a particular emphasis on assessing whether point_cloud_map table updating based on dissimilarity judgments proceeds as intended. In Section 4.3, to quantitatively assess whether the data transmitted from the vehicle are processed without delay, we evaluate distributed processing capability, data compression rate, and total processing time. We also evaluate performance under the same scenario for the conventional relational database (RDB) cluster and compare them with our proposed system.

4.1. Evaluation Scenario

The primary objective of our system is an architecture that effectively detects changes in point cloud data in real-time and updates the point cloud map accordingly. To determine whether this feature operates correctly, we conducted an actual driving test with multiple replicate drives on a specific route depicted in Figure 7. The experimental scenario for the qualitative evaluation was as follows.

Figure 7. Experimental route at Konkuk University.

1. After the first drive, extract the GPS data from the gps_position_log table and the point data from the point_cloud_map table into CSV (comma-sorted values) files.
2. After the second drive, extract the data into CSV files.
3. Identify any changes in the road environment between the first and second drives. Determine the times and GPS coordinates at which these changes were observed.
4. Convert each CSV file into a PCD file and merge it with the GPS coordinates to create a point cloud map based on the ENU (east, north, up) coordinate system.
5. Visualize using Python and check whether the point cloud map at the location where changes were observed in step 3 has been updated.
6. To determine whether the point_cloud_map table has been appropriately updated, use spatial queries to check the data around the location where changes were observed.

Through the above process, we conducted a qualitative evaluation. For a quantitative evaluation, driving data were recorded to an ROS bag file during 20 drives. Subsequently, to verify data compression performance, we compared the row-based and columnar tables in both their distributed and non-distributed forms. We compared query response times by varying the number of nodes in the system to demonstrate the distributed processing capability among the nodes in the proposed system. Additionally, we measured the total processing time from the moment the data were collected until they were finally updated or dismissed. This measurement approach involved periodically purging the analyzed data from the point_cloud_analysis table to reduce unnecessary overhead. In conclusion, to

evaluate the effectiveness of the proposed architecture, we established a distributed cluster with a conventional relational database, as opposed to HTAP architecture, and conducted the same evaluation, comparing the results.

4.2. Qualitative Evaluation

Examples of changes detected during implementation of the scenario laid out in Section 4.1 are presented in Figure 8, with the left-hand panels representing the first drive and the right-hand panels representing the second drive. The upper portion of Figure 8 displays an image captured by a monologue camera, while the lower portion depicts an image captured by the LiDAR sensor. The data captured by the LiDAR sensor were visualized using the Ros Rviz software version melodic on the Ubuntu operating system. The operation of the architecture was verified by targeting the vehicle observed during the second drive (highlighted in a red box).

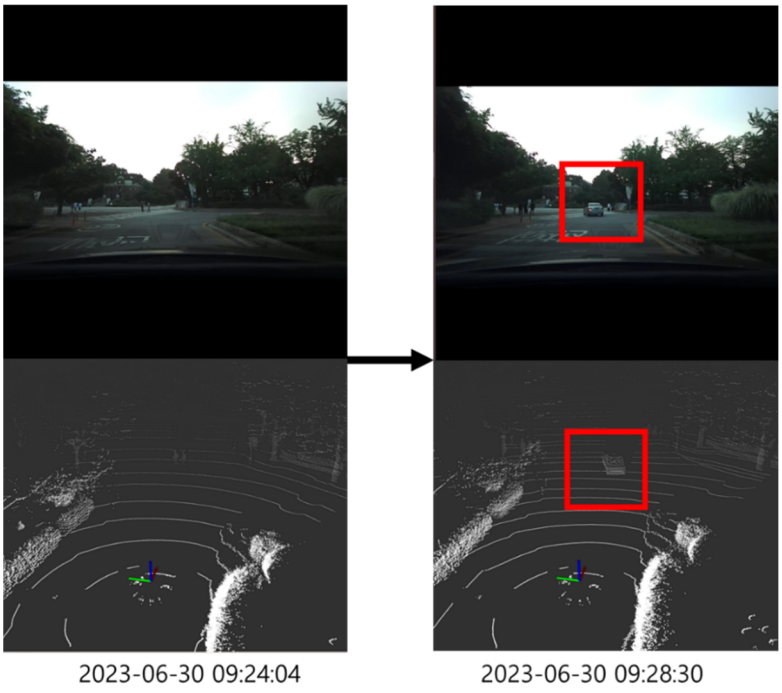

Figure 8. Detected changes. **Left**: first drive; **right**: second drive; **upper**: camera; **lower**: LiDAR.

Figure 9 compares the maps created from the point_cloud_map table values after each drive. The map from the first drive (left panel), during which no vehicle was detected, autonomously updated itself after detecting changes in the point cloud data due to the vehicle observation during the second drive. The updated map (right) shows the changes observed during the second drive (highlighted in a red box).

In this study, we focus on ensuring the currency of the original point cloud data for map creation, rather than on the creation of point cloud maps. Figure 9 visualizes the data from the point_cloud_map table, which was extracted into PCD files and merged without any additional processing, based on the ENU coordinate system, to verify if the data have been updated to their latest version. As seen in Figure 9 (right), moving objects appear as afterimages, indicating that the original data have been appropriately updated.

The vehicle was first detected at a longitude of 127.076675° E and a latitude of 37.5425422° N. Therefore, we examined the data from the point_cloud_map table around this location. Figure 10 compares the results obtained from querying the point_cloud_map table spatially.

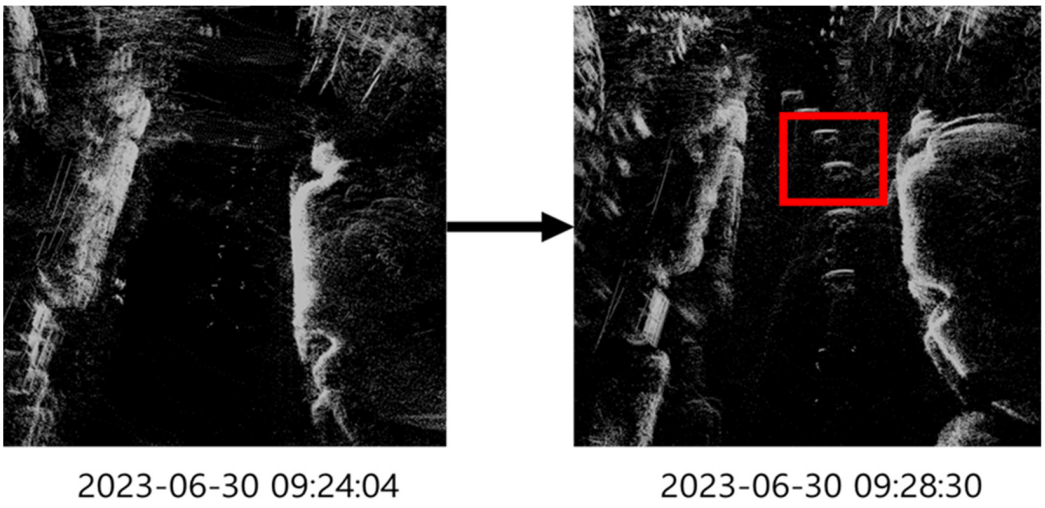

Figure 9. Point cloud map comparison. **Left**: before update; **right**: after update.

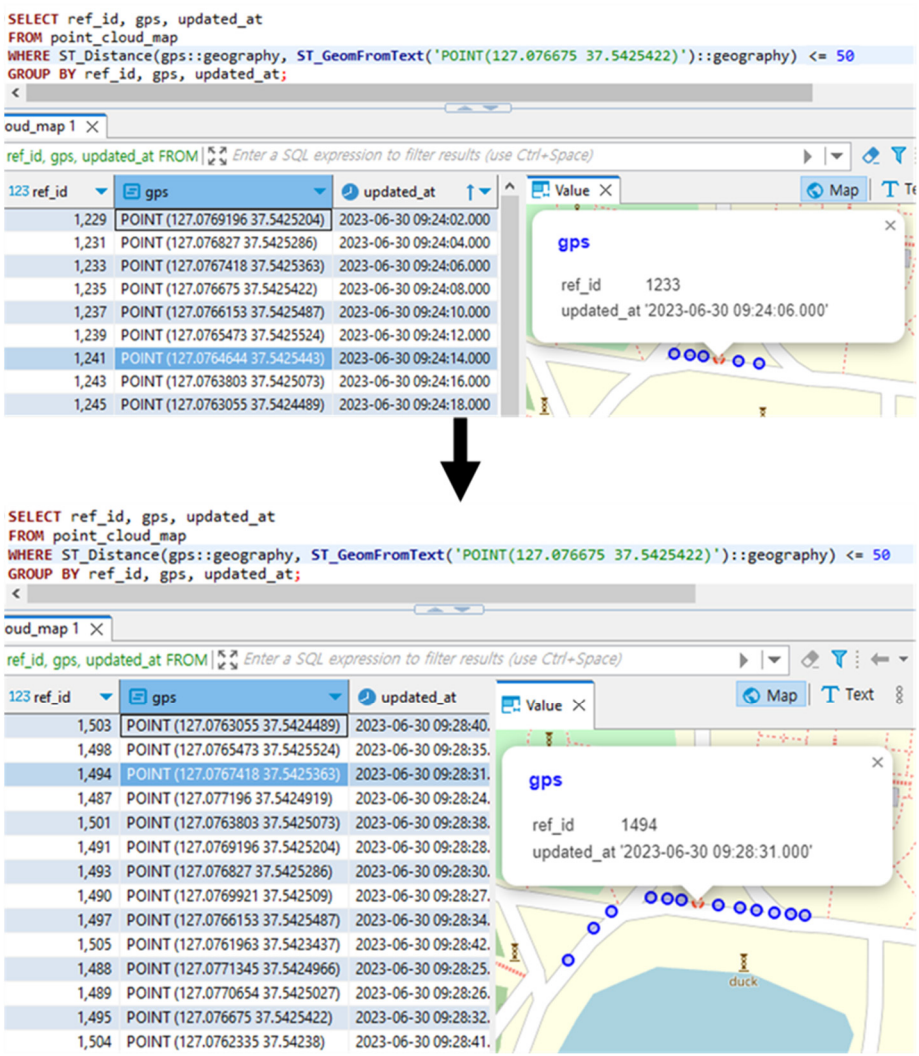

Figure 10. Spatial query results. **Upper**: before update; **lower**: after update.

To verify whether the point cloud data at the location where the vehicle was detected has been updated, we used a spatial query to search for GPS values within 50 m of the location where the vehicle was detected in the point_cloud_map table. After the first drive, the query result showed that the GPS data were updated at "2023-06-30 09:24:06". These

GPS data were confirmed to have been updated after the second drive at "2023-06-30 09:28:31", indicating that the architecture operated correctly.

4.3. Quantitative Evaluation

To evaluate the performance of the architecture, we assessed the compression rate by comparing our system's storage type with other storage types in terms of how the rate of storage size increased as the data volume increased. Additionally, we evaluated the system's response time for different types of queries as the data volume increased, and we evaluated its distributed processing capability as the number of nodes increased.

For the comparison of data compression rates, we conducted experiments with four different table types. The types of tables selected were a distributed columnar table, a distributed row-based table, a row-based table, and a columnar table. After each drive, the size of the table was measured. Figure 11 represents the measured data as a graph.

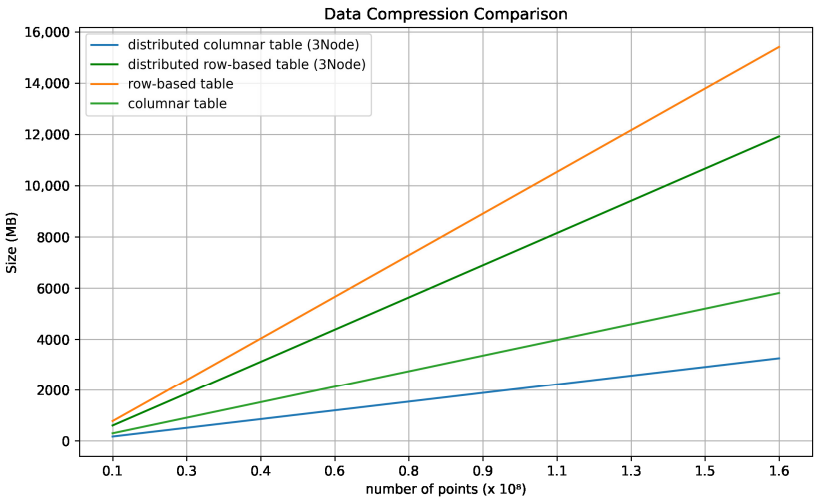

Figure 11. Data size by storage type according to the number of points.

The total number of points to be handled in 20 runs was approximately 160,000,000, and the distributed columnar table was the most compressed, with a size of 3254 MB, which is 21% of the size compared to the row-based table. In our architecture, the point_cloud_map table is continuously updated, thus restricting the overall data increase. However, the point_cloud_analysis table continues to accumulate for additional analysis. Therefore, we designed the point_cloud_analysis table as a distributed columnar table. Indeed, the point_cloud_analysis table was compressed to about 40% compared to the PCD files stored in S3.

Figure 12 presents a graph showing the changes in query response time as the total size of the stored data increased over the course of 20 drives while continuously operating the architecture. To evaluate the HTAP workload processing performance and distributed processing performance of the architecture, we differentiated the response times for OLTP and OLAP queries. We then conducted the same experiment after increasing the number of nodes in the architecture from 3 to 5.

Figure 12 displays consistent linear growth rather than a rapid spike. This behavior suggests that the architecture was running stably and handling the load predictably. When the architecture's node count was increased to five, the query response time growth rate was about 20% lower than with three nodes. This result indicates that the coordinator node was effectively managing distributed parallelization for both OLTP and OLAP queries. These findings show that the distributed processing feature of the architecture introduced in this study was functioning reliably.

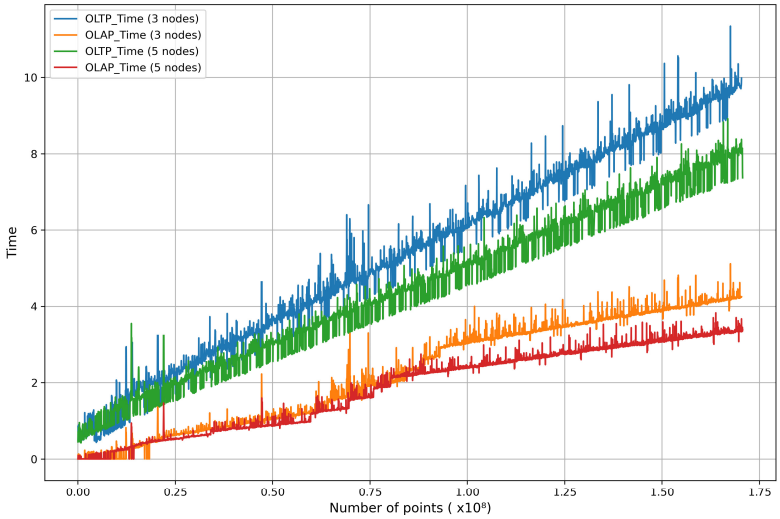

Figure 12. Response time by query type according to stored data size.

However, Figure 12 makes it clear that as the total size of the stored data increased, the system performance worsened. In the initial stages with a smaller stored data volume, the OLTP query response time was 0.86 s, and the OLAP query response time was about 0.11 s, resulting in a total response time of 0.97 s. However, as the stored data volume grew to around 160,000,000 entries, the OLTP query response time increased to 9.60 s, and the OLAP query response time reached approximately 4.28 s, summing up to 13.88 s in total. To achieve the system objective of updating point clouds in real-time, it is necessary to purge previously analyzed data. Therefore, we adopted a method of periodically purging already analyzed data from the point_cloud_analysis table to reduce unnecessary overhead. An evaluation conducted using this approach is shown in Figure 13.

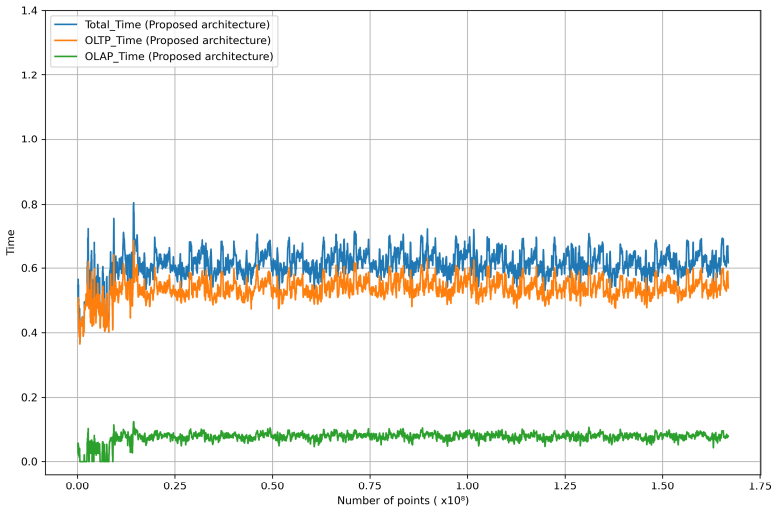

Figure 13. Response time by query type and total processing time of proposed HTAP architecture.

Figure 13 depicts response times for OLTP and OLAP queries and the total processing time from the moment data was collected until it was finally updated or dismissed in the proposed architecture. The proposed system demonstrates consistently stable performance metrics during repeated drives. The OLAP query response time averaged 0.07439 s, while the OLTP query response time was 0.52409 s, resulting in a total response time of 0.59848 s. Furthermore, the total processing time, including the time from data collection in the vehicle to preprocessing and transmission to the architecture, averaged 0.62476 s. Due to the combination of columnar tables and row-based tables in the proposed architecture, the response time for OLAP queries was especially outstanding, with less than 0.1 s. As we

receive data from the vehicle at a 1-s interval, we consider a real-time criterion to be within 1 s. Accordingly, the proposed architecture can update point cloud data within an average of 0.6476 s, thereby confirming its real-time capability.

We selected a distributed RDB cluster as the comparative group for the proposed architecture. The distributed RDB cluster was built using PostgreSQL, and all configurations were similar except that they were constructed using row-based storage only, without columnar storage. Figure 14 presents the results of the same evaluation conducted on the distributed RDB cluster, which exhibited overall unstable performance metrics. The average response time for OLTP queries was 0.46209 s, and for OLAP queries, it was 0.30399 s, resulting in an average total response time of 0.76608 s. Since it was constructed using row-based tables only, it performed OLTP query response times about 0.062 s faster than the proposed architecture. However, the OLAP query response time was approximately 0.22960 s higher than that of the proposed system, resulting in nearly a threefold difference. The average total processing time was 0.795412 s, which was approximately 27% higher than that of the proposed architecture. While it was measured within our real-time criteria of less than 1 s, there were numerous instances where it slightly exceeded or approached 1 s. This divergence was attributed to the OLAP query response time, indicating an unsuitable pattern for aggregation and analysis queries.

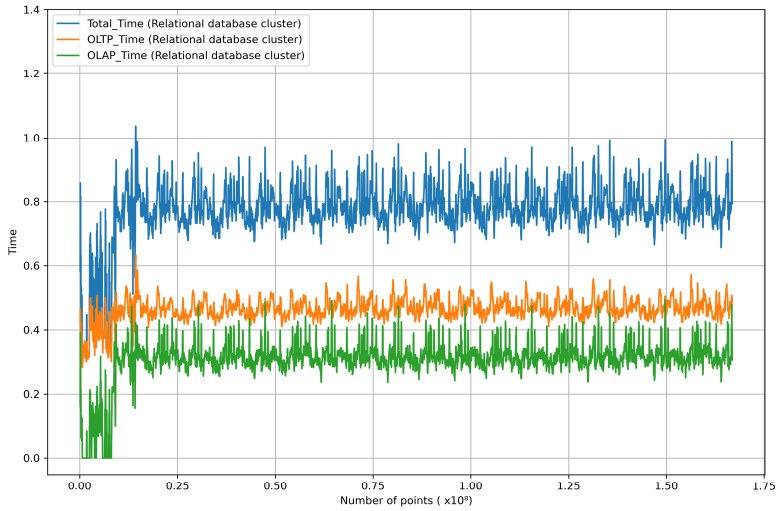

Figure 14. Response time by query type and total processing time of distributed RDB cluster.

5. Conclusions

In this study, we constructed and evaluated a HTAP (hybrid transactional/analytical processing) architecture that processes and updates point cloud data in real-time using distributed database technology. Unlike previous studies that managed LiDAR data for point cloud map construction in Hadoop, we built a HTAP architecture using a spatial database. This architecture allows real-time insertion, updating, and analysis of point cloud data. The architecture is divided into an Edge part operating in the ROS environment of the vehicle and a Cloud part operating in the PostgreSQL environment of cloud services. Data transmission between the two parts is facilitated through Kafka, and a data pipeline preprocesses point cloud data via a client on AWS Lambda and inserts it into the architecture.

The architecture developed in this study has a feature that autonomously detects changes over time by determining the dissimilarity between sequential runs through the environment and maintaining and managing the latest state of point cloud data. By offering this feature, the proposed system helps enable autonomous vehicles to generate safe, optimized routes. Through a qualitative evaluation of this feature, we confirmed that this feature operates correctly, updating the point cloud map as intended. In the quantitative evaluation, we demonstrated the real-time capability of our architecture with an average

total processing time of 0.6476 s, which meets our real-time criterion of a 1-s interval for data generation when continuous table management is performed. We compared the total processing time and processing stability between our proposed architecture and the distributed RDB cluster, confirming approximately 27% faster processing speed and a stable analysis pattern in favor of our architecture. Moreover, by observing faster processing times as the number of distributed nodes increased, we validated the parallel distributed processing capability for queries. While we identified a potential performance degradation with an absolute increase in data volume, we showed that real-time performance can be maintained by adding and redistributing nodes thanks to the parallel distributed processing capability.

However, the limitations of our study to be advanced in the future can be summarized as follows:

- The architecture was designed to handle point cloud data processing and updates for a single vehicle. Its performance under simultaneous analysis and updates from multiple vehicles is yet to be fully investigated.
- The proposed HTAP architecture demonstrates potential for consistent real-time performance. However, ensuring seamless scalability and robustness of the architecture for uninterrupted operation and expansion during runtime remains an area that requires further development.

We are currently investigating potential performance degradation when multiple vehicles simultaneously access the system for analysis and updates. Through this ongoing research, we anticipate constructing a more robust HTAP architecture.

Author Contributions: Conceptualization, J.K. and C.M.; methodology, J.K.; software, J.K.; validation, J.K.; formal analysis, J.K.; investigation, J.K.; resources, C.M.; data curation, J.K.; writing—original draft preparation, J.K.; writing—review and editing, J.K. and C.M.; visualization, J.K.; supervision, C.M.; project administration, J.K.; funding acquisition, C.M. All authors have read and agreed to the published version of the manuscript.

Funding: This paper was supported by the Korea Institute for the Advancement of Technology (KIAT) grant funded by the Korean Government (MOTIE) (P0020536, HRD Program for Industrial Innovation).

Data Availability Statement: Data is available from the authors.

Acknowledgments: This paper was written as part of Konkuk University's research support program for its faculty on sabbatical leave in 2023.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Rancho, Y. Overcoming the Challenges of 3D Mapping for Autonomous Cars. Available online: https://embeddedcomputing.com/application/automotive/adas-autonomous-drive/overcoming-the-challenges-of-3d-mapping-for-autonomous-cars (accessed on 8 August 2023).
2. Chris, T. What Lidar Is and Why It's Important for Autonomous Vehicles. Available online: https://www.autoweek.com/news/a36190274/what-lidar-is/ (accessed on 9 August 2023).
3. Apache Hadoop HDFS Architecture. Available online: https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html (accessed on 9 August 2023).
4. Stonebraker, M.; Daniel, J.; Abadi, A.B.; Chen, X.; Cherniack, M.; Ferreira, M.; Lau, E.; Lin, A.; Madden, S.; O'Neil, E.; et al. C-store: A column-oriented DBMS. In Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'05), Trondheim, Norway, 30 August–2 September 2005; VLDB Endowment: Los Angeles, CA, USA, 2005; pp. 553–564.
5. Wikipedia. Hybrid Transactional/Analytical Processing. Available online: https://en.wikipedia.org/wiki/Hybrid_transactional/analytical_processing (accessed on 9 August 2023).
6. Huang, D.; Liu, Q.; Cui, Q.; Fang, Z.; Ma, X.; Xu, F.; Shen, L.; Tang, L.; Zhou, Y.; Huang, M.; et al. TiDB: A raft-based HTAP database. *Proc. VLDB Endow.* **2020**, *13*, 3072–3084. [CrossRef]
7. Yang, J.; Rae, I.; Xu, J.; Shute, J.; Yuan, Z.; Lau, K.; Zeng, Q.; Zhao, X.; Ma, J.; Chen, Z.; et al. F1 lightning: HTAP as a service. *Proc. VLDB Endow.* **2020**, *13*, 3313–3325. [CrossRef]

8. Aunn, R.; Periklis, C.; Angelos, C.; Anastasia, A. Adaptive HTAP through Elastic Resource Scheduling. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), Portland, OR, USA, 14–19 June 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 2043–2054. [CrossRef]
9. Lyu, Z.; Zhang, H.H.; Xiong, G.; Guo, G.; Wang, H.; Chen, J.; Praveen, A.; Yang, Y.; Gao, X.; Agraval, A.; et al. Greenplum: A hybrid database for transactional and analytical workloads. In Proceedings of the 2021 International Conference on Management of Data, New York, NY, USA, 20–25 June 2021; pp. 2530–2542.
10. Vo, A.V.; Konda, N.; Chauhan, N.; Aljumaily, H.; Laefer, D.F. Lessons learned with laser scanning point cloud management in Hadoop Hbase. In Proceedings of the Advanced Computing Strategies for Engineering: 25th EG-ICE International Workshop 2018, Lausanne, Switzerland, 10–13 June 2018; Proceedings, Part I 25. Springer International Publishing: Berlin/Heidelberg, Germany, 2018; pp. 231–253.
11. Awrangjeb, M. Effective generation and update of a building map database through automatic building change detection from LiDAR point cloud data. *Remote Sens.* **2015**, *7*, 14119–14150. [CrossRef]
12. Lee, J.; Lee, K.; Yoo, A.; Moon, C. Design and Implementation of Edge-Fog-Cloud System through HD Map Generation from LiDAR Data of Autonomous Vehicles. *Electronics* **2020**, *9*, 2084. [CrossRef]
13. Lewis, P.; Mc Elhinney, C.P.M.; McCarthy, T. *LiDAR Data Management Pipeline; From Spatial Database Population to Web-Application Visualization*; Article 16; Association for Computing Machinery: New York, NY, USA, 2012; pp. 1–10.
14. Lee, K.; Hong, D.; Kim, J.; Cha, D.; Choi, H.; Moon, J.; Moon, C. Road-network-based event information system in a cooperative ITS Environment. *Electronics* **2023**, *12*, 2448. [CrossRef]
15. Ortega-Gomez, J.I.; Morales-Hernandez, L.A.; Cruz-Albarran, I.A. A Specialized Database for Autonomous Vehicles Based on the KITTI Vision Benchmark. *Electronics* **2023**, *12*, 3165. [CrossRef]
16. Kang, D.; Wong, A.; Lee, B.; Kim, J. Real-Time Semantic Segmentation of 3D Point Cloud for Autonomous Driving. *Electronics* **2021**, *10*, 1960. [CrossRef]
17. ROS.org—ROS/Concepts. Available online: http://wiki.ros.org/ROS/Concepts (accessed on 9 August 2023).
18. ROS.org—ROS/pcl_ros. Available online: http://wiki.ros.org/pcl_ros/Tutorials/VoxelGrid%20filtering (accessed on 9 August 2023).
19. Diaz, I. Ros-Kafka-Connector. 2019. Available online: https://github.com/isakdiaz/ros-kafka-connector (accessed on 4 September 2023).
20. Amazon Web Service Lambda. What Is AWS Lambda. Available online: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html (accessed on 4 September 2023).
21. PostreSQL. PostgreSQL(v15.2). Available online: http://www.postgresql.org/ (accessed on 4 September 2023).
22. PDAL. What Is PDAL? Available online: https://pdal.io/en/latest/about.html#what-is-pdal (accessed on 4 September 2023).
23. GiST Indexes. Available online: https://www.postgresql.org/docs/15/gist.html (accessed on 4 September 2023).