ISSN: 2278-4632
Vol-12 Issue-07 No.03 July 2022

Smart Vehicle Detection in Real-Time and Tracking System of the Vehicle Using Road Surveillance Cameras

Saswati Sahoo [1], Rakesh Muduli [2], Ipsita Sahoo [3], Abhishek Das [4], Rakesh Muduli [5], Sarbajit Panda [6]
BHABANISANKAR SAHANI[7]
[1, 2, 3, 4, 5, 6] Gandhi Institute for Education & Technology, Baniatangi, Khordha, Odisha
[7]NM Institute of Engineering & Technology,Bhubaneswar,Odisha
saswatisahoo@giet.edu.in, rakeshmuduli@giet.edu.in, ipsitasahoo@giet.edu.in, abhisekdas@giet.edu.in, rakeshmuduli@giet.edu.in, sarbajitpanda@giet.edu.in

Abstract

Numerous research have been done on the efficient implementation of security and surveillance based on video analysis as a range of video surveillance equipment including CCTV, drones, and automobile dashboard cameras have grown in popularity. The most difficult task in automobile-related surveillance, in particular, is car tracking. Analyzing frames from many video sources independently was one early method for completing such a task. The results from the analysis of a single video source are extremely constrained given the shooting distance of the majority of video devices. A collection of video sources should be taken into consideration in order to gain more thorough information for automobile tacking. The pertinent data should then be combined according on the geographical and temporal restrictions. Due to this, we suggest in this study a real-time vehicle monitoring system based on surveillance footage from various devices, such as CCTV, dashboard cameras, and drones. Our system includes a Frame Distributor, a Feature Extractor, and an Information Manager and is built on a distributed processing foundation for scalability and fault tolerance. The distribution of video frames to the processing nodes from diverse devices is the responsibility of the frame distributor. The Feature Extractor gathers key vehicle characteristics from each frame, including the licence plate number, time, and position. The Information Manager gathers pertinent data from the feature database to respond to user queries after storing all the features in a database. We put into practise a prototype system and ran several experiments to demonstrate the efficacy of our suggested solution. We report some of the results.

Keywords: Automobile tracking system, Real-time, Computer vision, Database, Index structure

1 Introduction

With the rapid advancement of IT technology, a number of video surveillance devices have entered a wide use for surveillance and security purposes in daily life. As a typical example, closed-circuit television (CCTV), also known as video surveillance, uses video cameras to transmit video signals to a limited set of monitors. When CCTV was first introduced, its poor quality and significant installation costs limited its applicability. Recently, because of improved definition, better distribution rates, and various basic functions of CCTV, more diverse applications have become easily implemented [1, 2]. Another popular example is the dashboard camera, car DVR or car black box, that is one or a pair of onboard cameras that continuously record (loop recording) the view through the windscreen. Dashboard cameras can provide video evidence in the event of a road accident or vandalism. For this reason, numerous cars are now equipped with dashboard cameras and, in a number of countries, dashboard cameras are mandatory on public transportation, such as buses and taxis.

While CCTV and dashboard cameras play similar roles, there is a significant difference between them, namely mobility. A CCTV is typically installed for sur- veillance in areas that require monitoring, such as banks and hospitals or areas where security is required. There- fore, its coverage is limited. On the other hand, as a car dashboard camera is installed inside a car, it can record while the car is moving. To perform car tracking efficiently, these two types of devices should be considered together. In the case of CCTV, as its location is fixed and its hardware performance is superb, it is highly effective for the monitoring of car movements in a predefined area. On the other hand, car dashboard cameras can cover a broad area including areas where CCTV is not appropriate. There could be areas not covered by both car dashboard cameras and CCTV.

* Correspondence: saswatisahoo@giet.edu.in

ISSN: 2278-4632
Vol-12 Issue-07 No.03 July 2022

Drone-mounted cameras, which have been attracting much attention recently, can be used effectively to cover such areas.

Numerous studies have been conducted to date using a number of video surveillance devices for, amongst others, traffic condition analysis [3], people identification [4], and event detection [5]. As they are dealing with a single video source, the analysis results were limited, and combining the results from separate video sources would be both time-consuming and labor-intensive. Car tracking based on surveillance videos suffers from the same problem. To solve this, in this study, we propose a Kafka-based real-time car tracking system that can col- lect data from different video sources, extract relevant features from cars for monitoring, and integrate them in a consistent manner. Our system is designed to utilize various widespread devices, including CCTV and dashboard cameras, as the effectiveness of car tracking relies on diverse information, such as plate number, time, place, and direction, collected from numerous different places. Fortunately, modern CCTV, drones, and dashboard cameras provide diverse metadata including global positioning system (GPS) and timestamping. In addition, plate number and moving direction can be easily detected from the captured images using popular image processing or machine learning techniques.

Our system can be used effectively to handle traditional surveillance tasks that are typically both time-consuming and labor-intensive. For instance, one of the typical steps for the police to determine the movement of a stolen vehicle is to start with the CCTV and dashboard cameras in the vicinity and gradually expand to a greater area. Investigating all the CCTV records and dashboard cameras involved would require significant amounts of human labor and time. In the case of our system, based on the car plate number, time of the crime, and place, we can easily formulate a query to determine the detailed track of the stolen car. In addition, our system can be highly effective for other popular applications such as traffic congestion analysis by region, searching for optimal driving routes, and planning new road construction.

However, in spite of the outstanding properties of our proposed system, it is not easy to implement for a number of reasons. Firstly, the system should have sufficient storage and processing capacity to handle the big data involved. The volume of data generated from the video devices in real time is significant. Therefore, the system should be sufficiently fast to avoid any data accumulation inside the node, otherwise, all the nodes in the system could experience a memory shortage and, in the worst case, the entire system might stop. To overcome this problem, a distributed processing platform can be used. Secondly, the system should have a fault tolerance ability that is essential for the system to provide accurate and complete car tracking information. This means that when a node fault or transmission fault occurs, the system should be able to recover from the fault. Thirdly, precise and fast image processing methods should be supported to efficiently extract all the critical information about the cars in the frames. Finally, to answer user queries promptly, the system should have methods for managing a significant amount of data efficiently, including an index structure for query processing.

In this study, based on these investigations, we propose a real-time car tracking system IVATS (integrated video-based automobile tracking system) that can collect video big data, extract and store principal vehicle features, and process user queries in a real-time environment. Our proposed system comprises three components: Frame Distributor (FD), Feature Extractor (FE), and Information Manager (IM). The role of the FD is to assign a significant amount of frames from numerous video sources to processing nodes using Apache Kafka [6]. Each node in the FE extracts principal vehicle features such as plate number, time, and location from the frame and transfers them to the IM. The IM that is built on HBase [7] clusters is responsible for storing all the extracted features, constructing index structures for them, and retrieving all the relevant data to answer user queries.

The structure of this study is as follows. Section 2 describes a number of related studies and background information. Section 3 presents the overall structure of our proposed system. Section 4 describes the experiments that were performed, and Section 5 concludes this study.

2 Related works and background

Before we describe our system in detail, we introduce a number of related studies. We first investigate methods for recognizing or tracking automobiles from video frames, and then, we investigate frameworks for real-time distributed processing, distributed databases, and index structures for HBase.

2.1 Automobile recognizing and tracking

For automobile tracking based on surveillance video, it is essential to extract the primary vehicle features such as plate number, color, and size from a video frame. Nam et al. [8] classified the types of vehicles as, amongst others, SUVs, sedans, and RVs using images from visible light and thermal cameras. Suryatali et al. [9] reported a scheme for determining the direction and size of automobiles using Kalman filters. Solanki et al. [10] proposed a scheme for recognizing plate numbers by locating the plate number, segmenting character areas, and utilizing optical character recognition (OCR). Tarigan et al. [11] proposed a similar scheme using neural networks and genetic algorithms.

ISSN: 2278-4632
Vol-12 Issue-07 No.03 July 2022

As numerous diverse vehicle feature extraction methods have been developed, complete systems for vehicle tracking have also been proposed. In [12], Rao proposed a system that collected the frames from surveillance videos, recognized the license plate, and provided the results to the user that consequently enabled remote monitoring. Chen et al. [13] proposed a video surveillance system in a cloud environment. Because of the automatic license plate recognition engine and the cloud environment, their system was able to cover wide areas and visualized the detection results using Google Maps [14].

2.2 Frameworks for real-time distributed processing

Numerous recent approaches for real-time distributed processing are based on Hadoop [15] and Spark [16]. Hadoop [15] is a popular framework that makes it possible to process large data sets in a distributed environment and a number of studies, such as [17–19], used Hadoop for distributed image processing. In particular, in [18], a Hadoop image processing interface (HIPI) [19] was implemented based on MapReduce to handle large image sets. As Hadoop processes data in batches, HIPI is not appropriate for real-time processing. In addition, the Hadoop distributed file system uses a random-access approach to disks, which induces an amount of delay in accessing the data in a file system.

Spark [16] is another well-known framework suitable for distributed processing. The data structure, known as a resilient distributed dataset and memory-based processing, makes Spark one of the fastest frameworks. However, it has a critical weakness with insufficient memory. When it encounters insufficient memory, the processing speed of the system decreases rapidly and could even result in the data in the memory being lost.

The abovementioned disadvantages of the two popular frameworks could be significant stumbling blocks for real-time vehicle tracking. Therefore, we focus on Kafka [6], which is a platform developed for real-time message transmission. Kafka comprises three parts: Producer, Consumer, and Broker. Producer generates data and sends them to the Broker. In Broker, the data are classified according to their topics and replicated for increased reliability. Consumer, a processing part, obtains the data from Broker each time it finishes tasks.

Kafka has the following properties: it stores temporary data in its own file system, and each Consumer schedules its own task. Saving data in the storage nodes enables Kafka to recover the data without data loss when an error occurs. Although memory-based structures are typically faster than disk-based structures, the speed of data access in Kafka is comparable to that of memory-based structures because of efficient disk usage [20]. The second property indicates that a Kafka node need not wait for a job schedule from the cluster master. Therefore, bottleneck problems caused by scheduling can be avoided and the communication between nodes can be decreased, reducing the network load. Because of these properties, Kafka can be a suitable framework in a real-time environment, and it was validated in [21].

2.3 Distributed databases and index structures

Because of the popularity of distributed processing frameworks, distributed databases like Cassandra [22], MongoDB [23], and HBase [7] are attracting increasing attention for managing large volumes of data. Apache Cassandra [22] is an open-source distributed NoSQL database management system. Because of its decentralized structure, it can avoid bottlenecks caused by a master node. In addition, its performance increases proportionally with the number of nodes.

MongoDB [23] is another open-source cross-platform NoSQL database program. It is a categorized document-based database, while Cassandra and HBase are column-based databases. Compared to other database management systems, it is easy to use and can process a number of query conditions.

HBase [7] is an open-source, non-relational database based on Hadoop and Google Bigtable [24]. It ensures data consistency and provides fault tolerance. In addition, as HBase is based on Hadoop, it is easy to use MapReduce when implementing the various query processing methods. For this reason, we use HBase for data management. In HBase, a data tuple is called a row and data are managed in tables that are divided into small row sets known as region. Therefore, MapReduce performs data processing in the unit of region. Except for Rowkey, which is an identifier of a row, the attributes of a table are not indexed. This means that HBase must access all stored data to answer user queries. Therefore, data retrieval takes significantly longer than data insertion and the query processing time increases rapidly as the volume of stored data increases. To overcome this problem, a number of studies have proposed the index structure, specifically for geometric information.

A popular index structure for geometric information is R-tree [25]. Wang et al. [26] proposed an R-tree-based indexing scheme for trajectory data of cars in a distributed environment, and Du et al. [27] proposed an index structure with a number of R-trees and Hilbert space-filling curves [28]. Another index structure for geometric information is Quad-tree [29]. Chen et al. [30] indexed GPS data using Quad-tree and Hilbert space-filling curves, and Xie et al. [31] uti- lized HBase tables as an index based on Quad-tree. Inthis study, we optimized the index structure in [30] to obtain improved performance.

ISSN: 2278-4632
Vol-12 Issue-07 No.03 July 2022

3 Methods

In this chapter, we describe the overall structure of our IVATS for real-time car tracking in detail. The system comprises three parts: FD, FE, and IM. The FD is responsible for reliable distribution of the video frames from a number of devices to the processing nodes, FE extracts diverse vehicle features such as plate number, time, and location from each frame, and IM stores all the extracted feature data, processes user queries by collecting relevant information from the feature database, and presents the query results to the user. Figure 1 shows the structure of our system.

3.1 Frame distributor

For effective car tracking, we utilize multiple video sources including CCTV, drone-mounted cameras, and car dashboard cameras. The role of FD is transferring frames from diverse video sources to FE for feature extraction. One critical task of this module for accurate car tracking is reliable data transfer. As frames are generated from diverse video devices in real time, the data volume is significant and reliable data transfer is not trivial [32]. In addition, the frames must be processed rapidly, or the entire system could stop because of buffer overflow. To prevent this, we use a Kafka cluster for frame distribution and storage.

Stream Manager (SM) in FD divides the data from the stream channels into frames and transmits them to the Kafka cluster. The stream channels can be directly connected to a video device or receive data from a remote video device using a real-time protocol, such as the real-time streaming protocol. SM is responsible for either one stream channel or multiple stream channels, depending on its capacity. Each frame has three RGB (red, green, and blue) color channels, and SM transforms the frame into a byte array by serialization. The serialized byte array will be restored to its original form in FE for image processing. Therefore, SM must provide the metadata, including the width, height, and the number of color channels, to the FE node through the Kafka cluster. However, sending this information each time can result in a significant overhead. To reduce this overhead, the frame metadata for reconstructing frames is only sent once when the connection between SM and the Kafka cluster is created, and the Kafka cluster records this information for the FE. Frame-related metadata such as GPS data and time are sent in a byte array. As we are unable to get the exact data of the captured automobile, the GPS data and time refer to the location and time of capture of the video devices, and not the automobile. For instance, the GPS data in

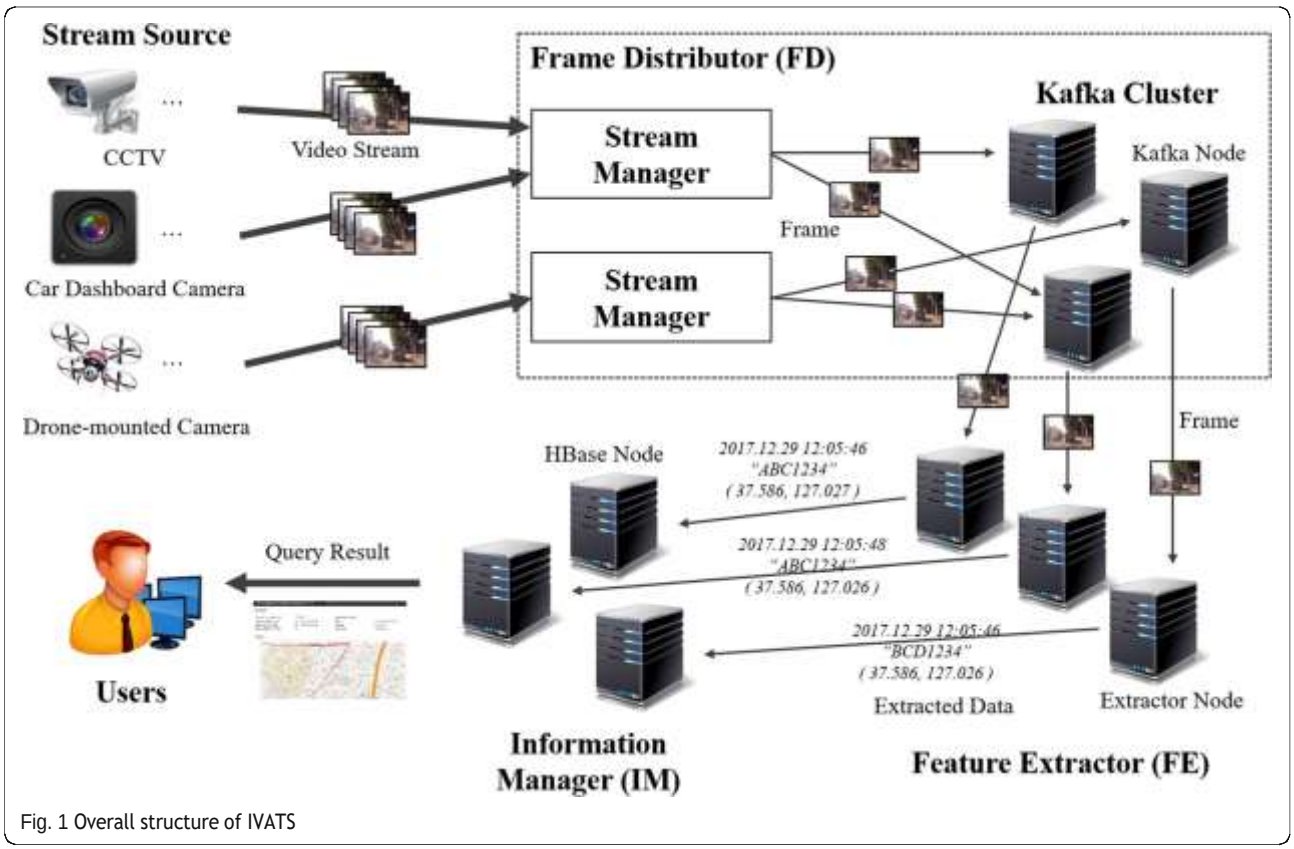

Fig. 1 Overall structure of IVATS

ISSN: 2278-4632
Vol-12 Issue-07 No.03 July 2022

the CCTV video is not identical to the vehicle data in the video. However, this difference is not a significant problem for car tracking in an actual environment.

Figure 2 shows the transmission steps between SM and a node in a Kafka cluster. As soon as a connection is made between SM and a node, SM sends the metadata needed for image reconstruction and starts to send frames by converting each frame into a serialized byte array, bytearray$_t$ in the figure, at time t. $bytearray_t$ consists of RGB and I, which are the byte arrays of three color channels and the metadata respectively. $bytearray_t$ at a Kafka node will be sent to a node in the next step when the node requires the data.

While the FD is responsible for preparing frames tosend and distributing them, the Kafka cluster actually connects the stream channels to the FE. The Kafka clus- ter receives a byte array from SM and forwards it to the FE. The frame then becomes located in a Kafka topic, which is a message queue in Kafka. The FE nodes take a frame from Kafka topics each time they finish a task.

As discussed above, Kafka is superior to other frameworks such as Hadoop and Spark in a number of aspects. Firstly, the nodes in a Kafka cluster store the data in their local file system. Therefore, the data being transmitted in the Kafka cluster are always protected from data loss, even when a fault occurs. In addition, because of this property, Kafka can play a role as a data buffer and restore the original data without any loss when a node is at a standstill. The data can be deleted only when the data duration exceeds some predefined threshold. Secondly, similar to other frameworks, the Kafka Cluster also adopts replication for the situation when a number of nodes stop abruptly. These two properties give Kafka high availability. Thirdly, Kafka nodes can schedule their own tasks. Because of this, Kafka does not experience the overhead problems that occur when the master node must schedule all the slave nodes. Fourthly, Kafka nodes can be operated asynchronously. Synchronous operations are typically safe from errors caused by an incorrect processing order, although they

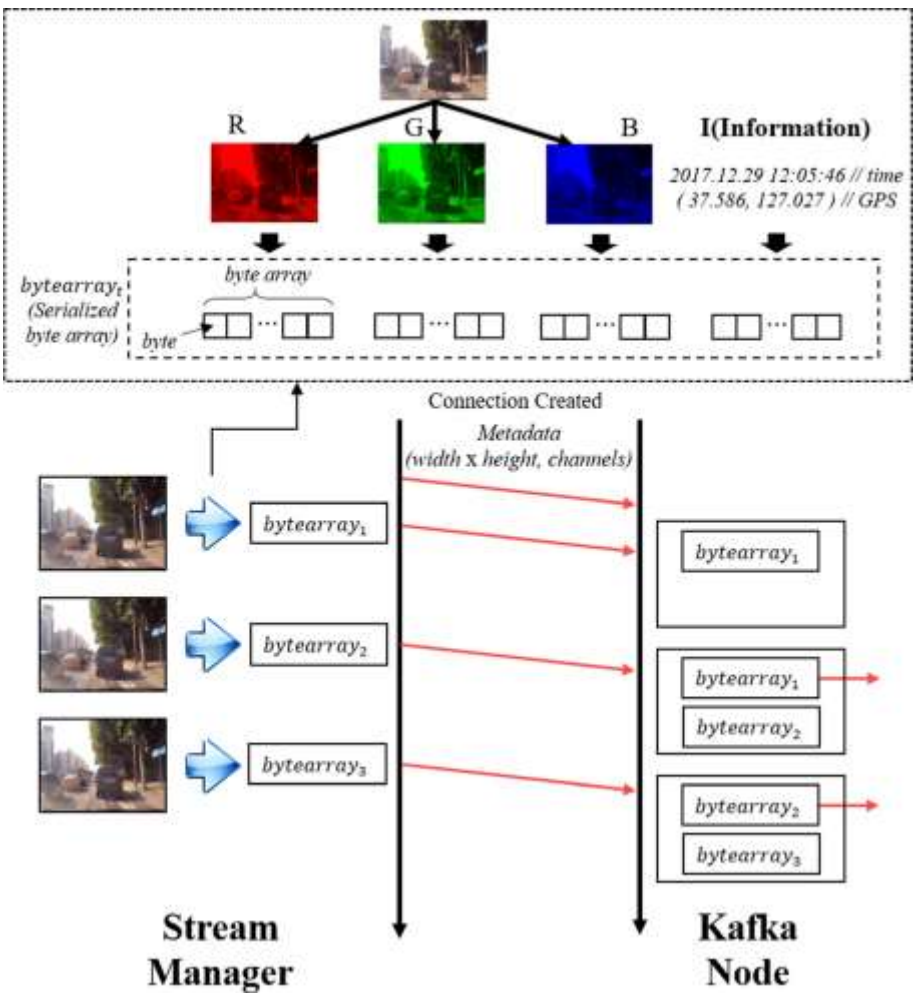

Fig. 2 Transmission steps between Stream Manager and a node in a Kafka cluster

ISSN: 2278-4632
Vol-12 Issue-07 No.03 July 2022

are slower than asynchronous operations. However, Kafka supports asynchronous operations without errors and can accommodate data rapidly. Finally, an FE node takes the frames to process as soon as it finishes its current task. This policy frees Kafka from having to monitor the task of nodes for job scheduling.

3.2 Frame extractor

Frame Extractor extracts diverse features for car tracking from the frames in the Kafka cluster nodes through image and metadata processing. FE nodes obtain a frame from the FD whenever they complete their current task. When an FE node is finished with a frame, the extracted features are stored in IM together with the GPS data and timestamp that were sent with the frame. Although for simplicity, the plate number is used as the only feature of the vehicle in this study; additional features such as color, vehicle type, and moving direction can be used for more versatile car tracking.

The FE node receives frames in the form of both byte arrays and their metadata for restoration such as width, height, and the number of color channels. Based on this metadata, the node transforms the byte arrays into images and then performs feature extraction. Figure 3 shows the approximate steps for extracting car features. The first step in feature extraction is the removal of noise in the frame by transforming the image from RGB scale to grayscale, followed by Gaussian blurring. We then choose the region of interest (ROI) that contains the plate number of a vehicle. To do that, the grayscale image is converted into a binary image using Otsu's method [33] that reduces a grayscale image into a binary image, considering that the intra-class variance of two classes, white and black, should be minimal. When an optimal threshold is found, the frame is converted into a binary image using this threshold. For the binary image, we apply the top-hat filter that is one of the morphology operations. After that, we calculate candidate ROI regions by using templates. Based on various license plate templates, we search which region in the image contains a license plate. The selected regions become ROI. Now, we are ready to identify characters in the license plate region. Based on the identification, we can confirm that the region is a license plate.

To identify characters, pixels in the region of a license plate should be split into a number of areas that could possibly indicate single characters. However, the size of the ROI could be different depending on the distance and angle between the video device and the license plate. We use the affine transformation to solve this problem. This is a mapping function between two affine spaces with points, straight lines, and planes retained. Because of its property that maintains ratios of distances between points lying on a straight line, it can make ROIs have similar sizes and reduce the distortion introduced during the transformation of an image.

The next step for the plate number identification is to divide ROIs into character regions by projecting the pixels in each ROI onto a horizontal axis and counting the foreground pixels in the axis. Based on this, we can

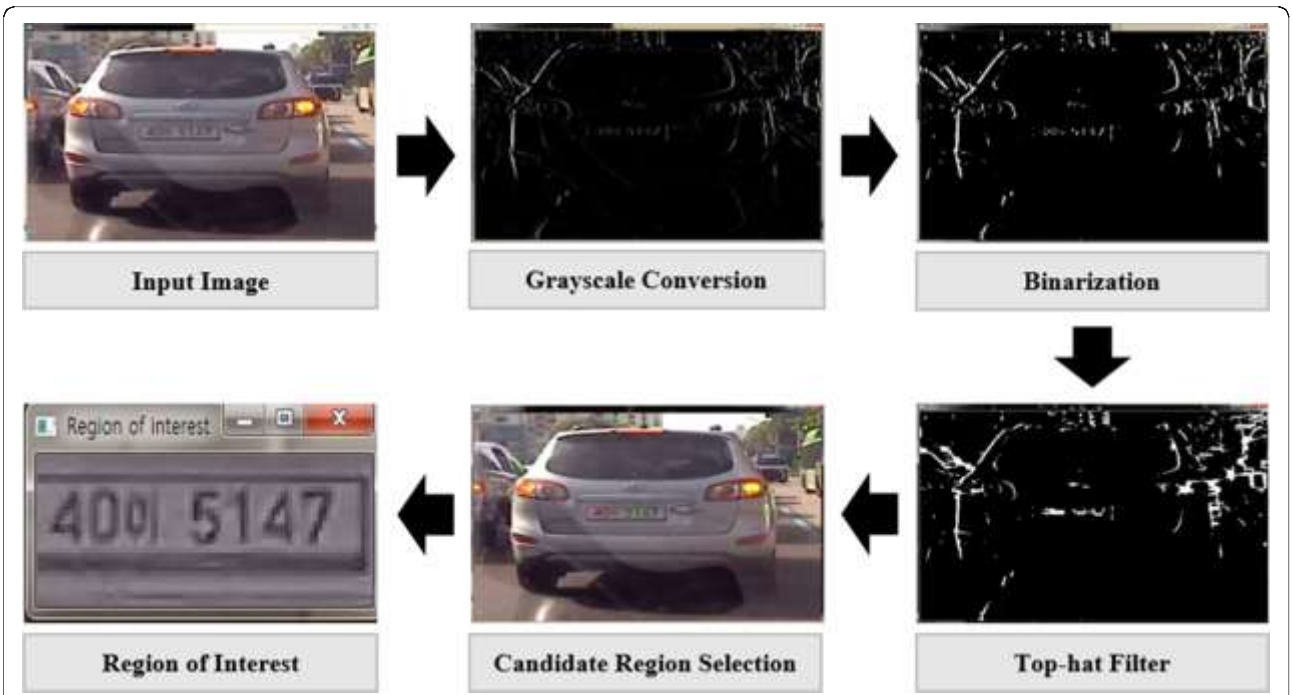

Fig. 3 Steps for identifying license plate

ISSN: 2278-4632
Vol-12 Issue-07 No.03 July 2022

Table 1 Sample HBase table schema

	ColumnFamily	
Rowkey	Latitude	Longitude
11 1111_1488960532177	37.58268	127.02621
11 1111_1488960532183	37.58274	127.02672
11 1111_1488960532199	37.58307	127.02813

construct a pixel histogram and use it to find character regions. If an interval has any pixels, this interval is considered as a character region; otherwise, the interval is deemed as a space between two adjacent characters. After all, if a ROI has the required number of character regions, it really corresponds to a license plate. The character in each character region can be read using Tesseract-OCR [34], an open-source OCR library.

3.3 Information manager

All the vehicle information including the license plate number, GPS data, and timestamp should be stored and processed efficiently to support diverse applications and user requests. For this, we use HBase, an open-source, non-relational database based on Hadoop and Google Bigtable. An index structure is required for the spatial data as spatial data should be continually retrieved for car tracking.

Table 1 presents a sample table schema for storing license plate number, time, and GPS data. We make the Rowkey of the table by combining the plate number and the time when the frame was captured and the other attributes are tied to one ColumnFamily that is a set of attributes in HBase. If additional attributes need to be stored, they will be contained in ColumnFamily in the current table, resulting in ColumnFamily extension. This enables HBase to simply append new rows instead of updating the previous row. The advantage of this is that there is no update overhead that is typically introduced in version management. In addition, searching for a specific vehicle data becomes considerably quicker as the data of an identical vehicle converges. In addition, considering that Rowkey is saved in each attribute, there is no requirement to create attribute columns for plate number and time and storage space could be decreased.

However, HBase cannot process queries with attributes other than Rowkey. Fortunately, this restriction can be overcome by using an index structure. Specifically, as latitude and longitude are two primary attributes in our system for representing geographical information, we can construct an index structure for the two attributes. We revised the method in [30] to meet our requirements. The index structure comprises two parts: R-tree [25] and Hilbert space-filling curve [28].

R-tree is one of the most popular data structures for spatial data indexing and has a number of versions [26, 27, 35–37]. The basis of R-tree is a rectangle binding a number of data points and minimizing its

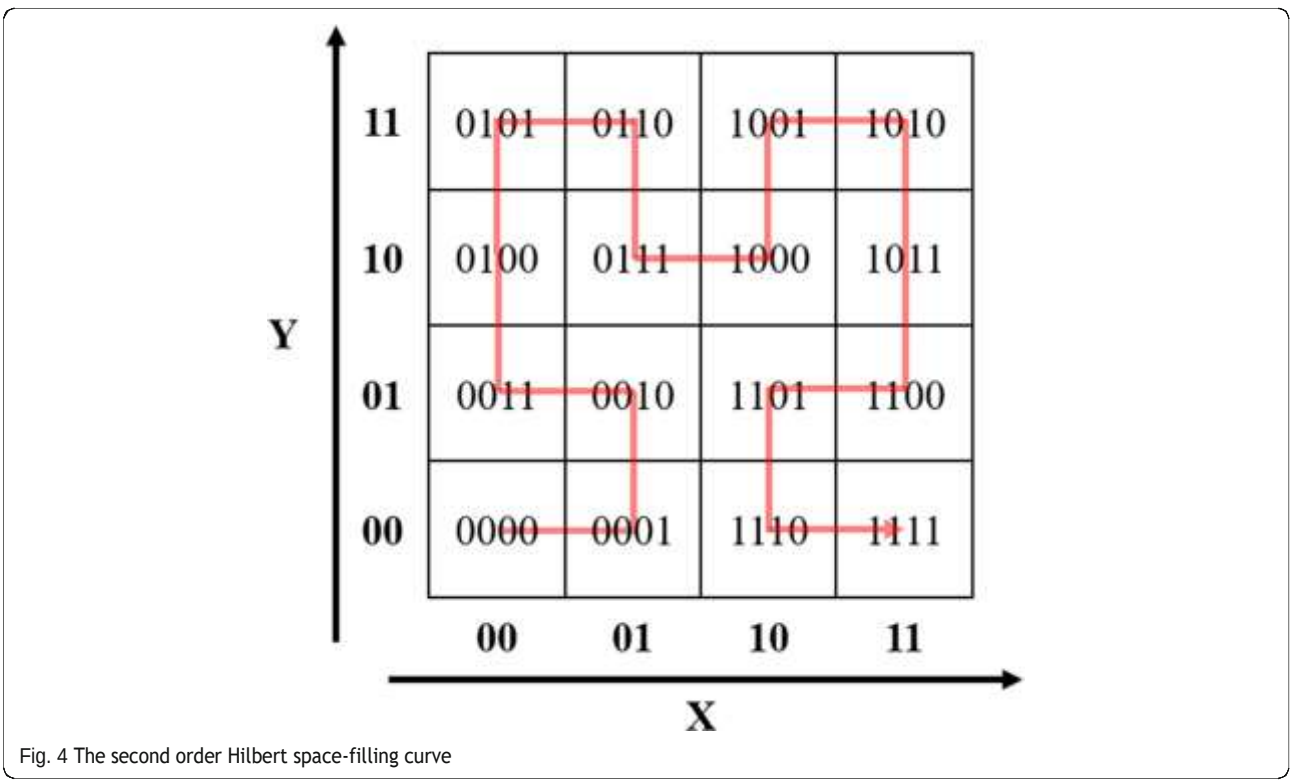

Fig. 4 The second order Hilbert space-filling curve

ISSN: 2278-4632
Vol-12 Issue-07 No.03 July 2022

boundaries that is known as the minimum bounding rectangle. The significant attribute of R-tree is that it is a height-balanced structure and provides stable performance regardless of data location. The greater the volume of data stored, the greater the height of the R-tree, and the longer the query response time becomes. To solve this problem, we spread R-tree using a space-filling curve.

A space-filling curve is a curve whose range contains the entire two-dimensional (2D) square. It is used to convert 2D data into one-dimensional (1D) data. The Hilbert space-filling curve, which is a fractal space-filling curve, exhibits the best performance in preserving locality [38, 39]. Because of this property, we use Hilbert space-filling curves in this study to map the spatial data (X, Y) to a 1D point. The total length of the Hilbert space-filling curve varies according to its order. As an example, Fig. 4 shows a second order Hilbert space-filling curve. The range of X and Y is from 0 to 3, respectively, in this case. The value in each rectangle is the value of the Hilbert curve corresponding to X and Y and the red line indicates the Hilbert curve line. In this curve, (3, 2) is converted to (1011) and other points, (0, 0), (3, 3), and (3, 1) are converted to (0000), (1010), and (1100), respectively.

Figure 5 shows the overall index structure for GPS data. Each node in HBase has an IndexManager that manages the index of all regions in the node. The IndexManager uses the two methods; Hilbert space-filling curve and R-tree. A Hilbert space-filling curve divides all possible spaces. Figure 5 shows a Hilbert space-filling curve of order 2. However, in the real application, Hilbert space-filling curve of order 7 is used to map actual GPS data. As the Hilbert space-filling curve does not use floating-point values, GPS data (latitude, longitude) should be converted into integer values. We divide all possible regions expressed by (latitude, longitude) to fit the Hilbert space-filling curve and map the regions into the areas in the curve. Thus, GPS data are allocated to some integer values of the curve and we use these values. Each independent area then has one R-tree whose nodes have latitude, longitude, and the name of the address where the data are stored.

In order to decrease the overhead of index updates because of the data insertion, we perform index updates only when HBase flushes all the data in the memory to the disk. Such flushing occurs just before the amount of data exceeds the memory capacity. This lazy update does not affect the performance significantly as the inserted data exists in the memory and data searching remains rapid, even if they are not indexed.

We now describe the detailed steps for processing spatial queries using a simple example. When a spatial query covering from (2, 2) to (3, 3) is given, its spatial condition is transformed into the range of a Hilbert space-filling curve from (1000) to (1011).

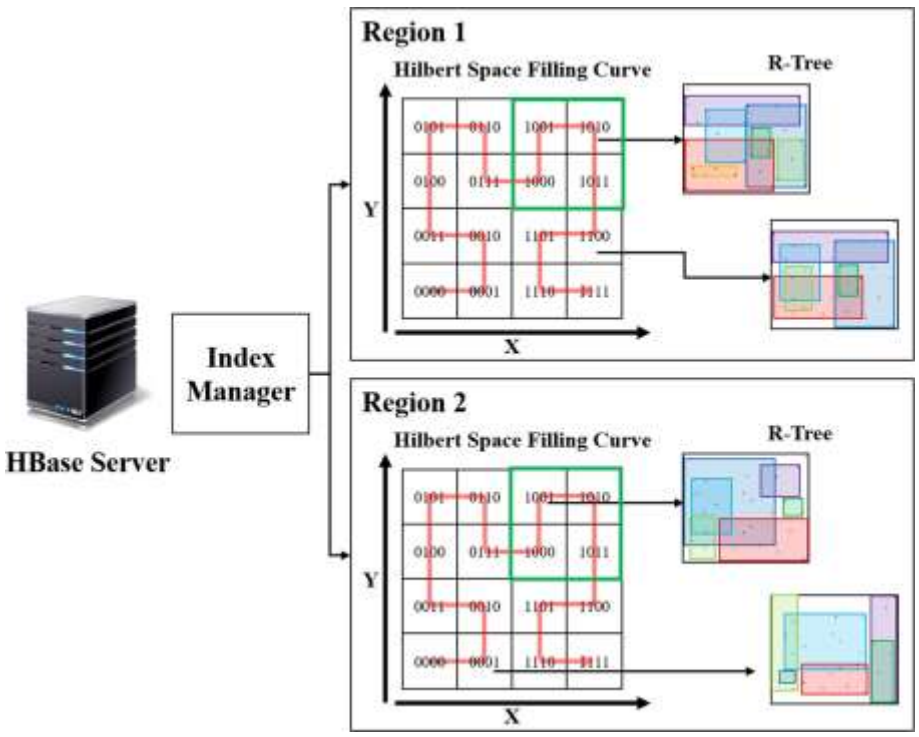

Fig. 5 Index structure

ISSN: 2278-4632
Vol-12 Issue-07 No.03 July 2022

(a)

{ "plate" : "40머5147",
"time" : "2016/08/05 09:01:44",
"latitude" : "37.637308",
"longitude" : "127.067590"}

(b)

{ "plate" : "80우2617",
"time": "2016/08/05 09:01:54",
"latitude" : "37.636116",
"longitude" : "127.068125" }

Fig. 6 Examples of feature extraction

This range is indicated as a green rectangle in Fig. 5. Then, we investigate each region whether an R-tree exists in the area contained in the range. For instance, in Fig. 5, the IndexManager in the figure has the index of two regions. Region 1 has R-trees in areas (1010) and (1100), and region 2 has R-trees in areas (0001) and (1001). Considering the query range, we can find that region 1 has an R-tree in (1010) and region 2 has an R-tree in (1001). If the area has an R-tree, the index structure retrieves the data address that is appropriate for the spatial condition. Accordingly, region 1 starts to search the R-tree in (1010) and collects the data addresses, and region 2 does the same for (1001). After all data addresses are collected through the R-tree searches, HBase records them as a result for users. For the complete query result, the

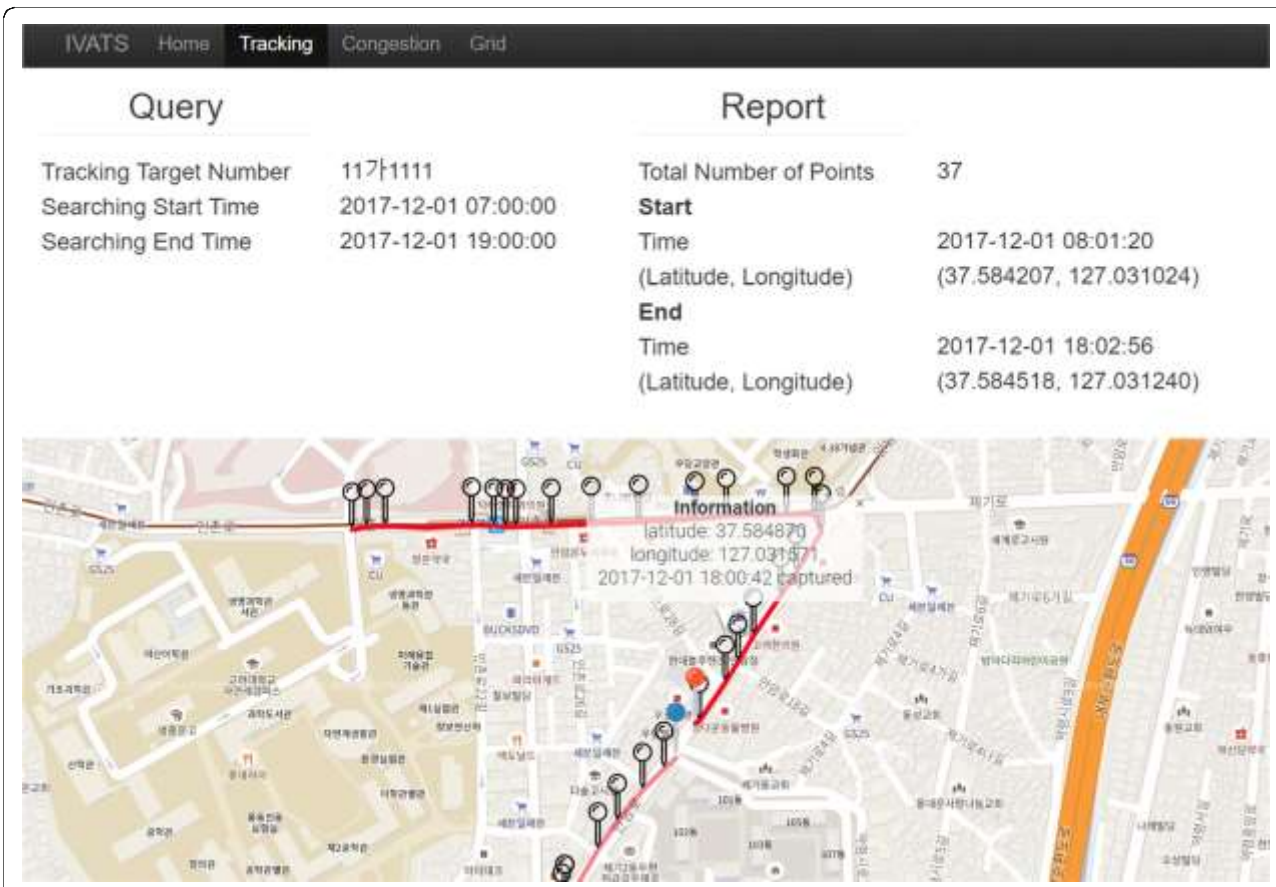

Fig. 7 An example of vehicle tracking result

ISSN: 2278-4632
Vol-12 Issue-07 No.03 July 2022

data in the memory should be checked because of the lazy update policy of our index structure. Therefore, spatial query results comprise data addresses from both the index and memory searches. For effective browsing of query results, they can be connected by means of a number of visualization tools.

4 Results and discussion

To demonstrate the performance of our proposed system, we performed a number of experiments. We also show how a number of typical car tracking queries are performed together with the query results. In the experiment, the FD was not considered as it was already reported in [40] that Kafka can be used as a satisfactory distributed-processing framework.

We consider the following three experiments: (1) extracting vehicle features, (2) visualizing query results, and (3) indexing spatial data. The experiments were performed on an Intel® Core™ i7-7700 with a 3.6 GHz processor and 32 GB RAM, using virtual machines running Ubuntu 16.04. In addition, we used Hadoop version 2.7.3 and HBase version 1.2.4. The number of nodes in the HBase cluster was five, of which one was the master and four were slave nodes. The data used in the experiments were virtually generated except for the data for the first experiment. Virtual data was used instead of actual data because the actual data available was insufficient for the needs of the tests. The virtual data generated for the experiments were 16 million tuples in the HBase table.

4.1 Vehicle feature extraction

Vehicle features of interest are extracted from frames by the FE node. Figure 6 shows two video frames from a car dashboard camera and their extracted features. In the features, the plate number is the outcome of the FE node and time, latitude, and longitude are transferred from the FD node with the frame. Even though we use the license plate number as a visual feature of a vehicle for simplicity in this work, it is easily extended to cover other visual features such as color, type of car, and direc- tion. The plate number in Fig. 6 is the candidate that has the greatest confidence among the extracted plate numbers. In a typical situation, the plate number of the vehicle immediately in front of the camera has the high- est confidence. When the plate numbers of other vehi-cles in the same frame are detected depending on the angle and distance, they can also be used for more com- prehensive car tracking. Lastly, all the collected features

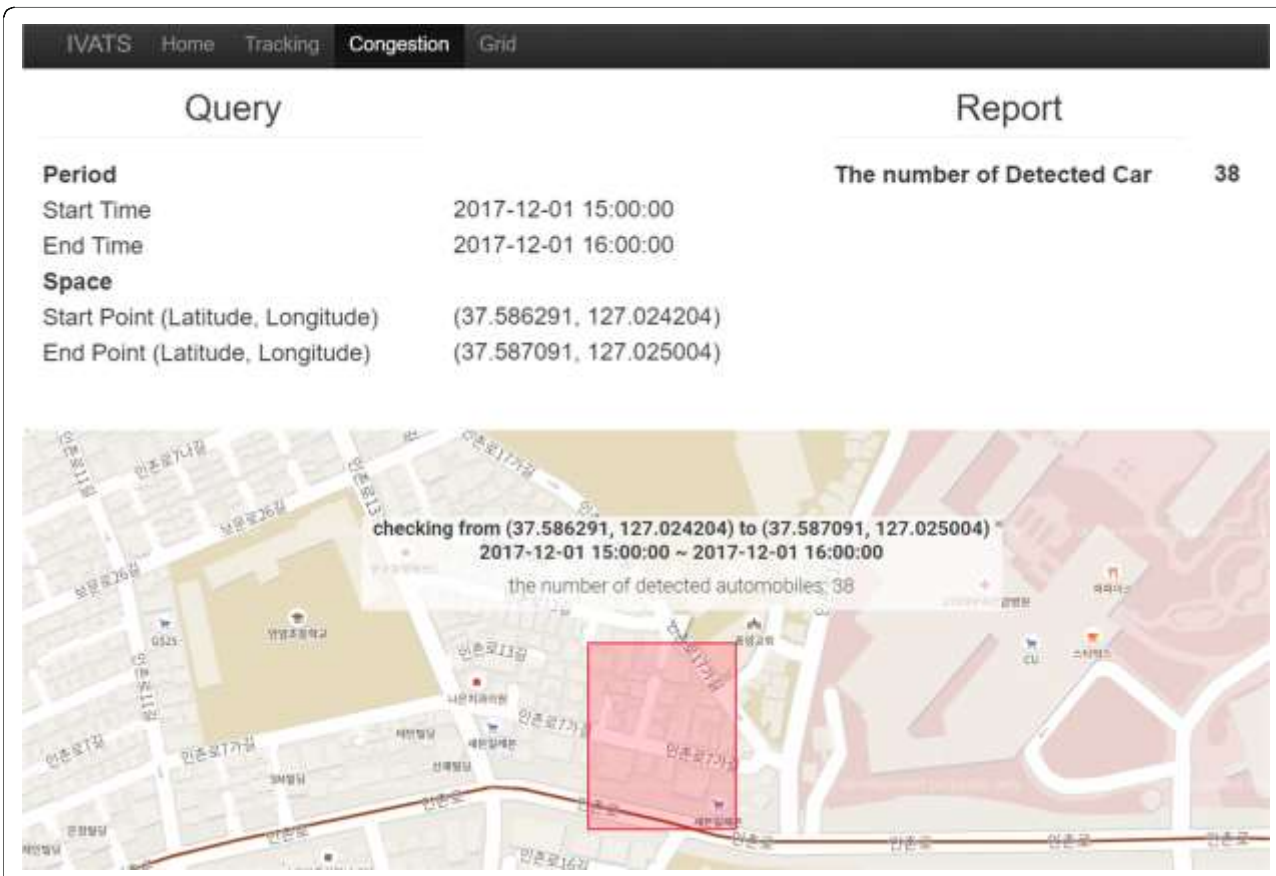

Fig. 8 An example of traffic evaluation in a specific region

ISSN: 2278-4632
Vol-12 Issue-07 No.03 July 2022

through FD and FE nodes are sent to IM for storing into the database.

4.2 Visualization

In the car tracking application, visualization could be an effective way for users to easily understand the query results. For this reason, we incorporated simple visualization functions in our system. Using these functions, the trajectory of a specific vehicle can be seen by using the data in IVATS, and the other vehicles can be seen sequentially. The trajectory of a vehicle is displayed on the map using Google Maps API [14].

For example, given a plate number and possibly temporal or spatial condition, the system visualizes all the records that satisfy the query condition in both the spatial and temporal order. Figure 7 shows the result of tracking a particular vehicle. The result consists of two parts. The top part of the result shows the summary of the requested query and the query result including the first and last locations of the vehicle captured by IVATS and the number of times the vehicle was seen. The bottom part shows the trajectory of the vehicle on the map. Black markers on the map indicate the locations where the vehicle was captured. In particular, the first and last locations of the tracking are marked in blue and red, respectively. The red line connects all the locations where the vehicle moved along. In fact, the line represents the trajectory of the vehicle. The pop-up window shows the location and time information of the selected marker so that users can browse the trace of the vehicle. Our proposed system can handle his kind of user query easily considering our policy for constructing and storing Rowkey for table tuples.

Even when the user query has a particular time range or a specific area of a rectangle expressed by two points, our system can easily give an answer to the query. This type of query is highly effective for measuring the traffic congestion in a specific area. Figure 8 shows an example. In the figure, the top part shows the user query, which contains the time range, area of interest, and the number of vehicles detected. The bottom part shows a map where the given query range is represented by a red box and the query result is shown on the pop-up window. To process such queries accurately, matched tuples should be sorted by the plate numbers as a vehicle could appear in numerous frames from different video sources. For instance, if a vehicle A was captured by vehicles B and C in rapid succession, there would be duplicate data in the query result.

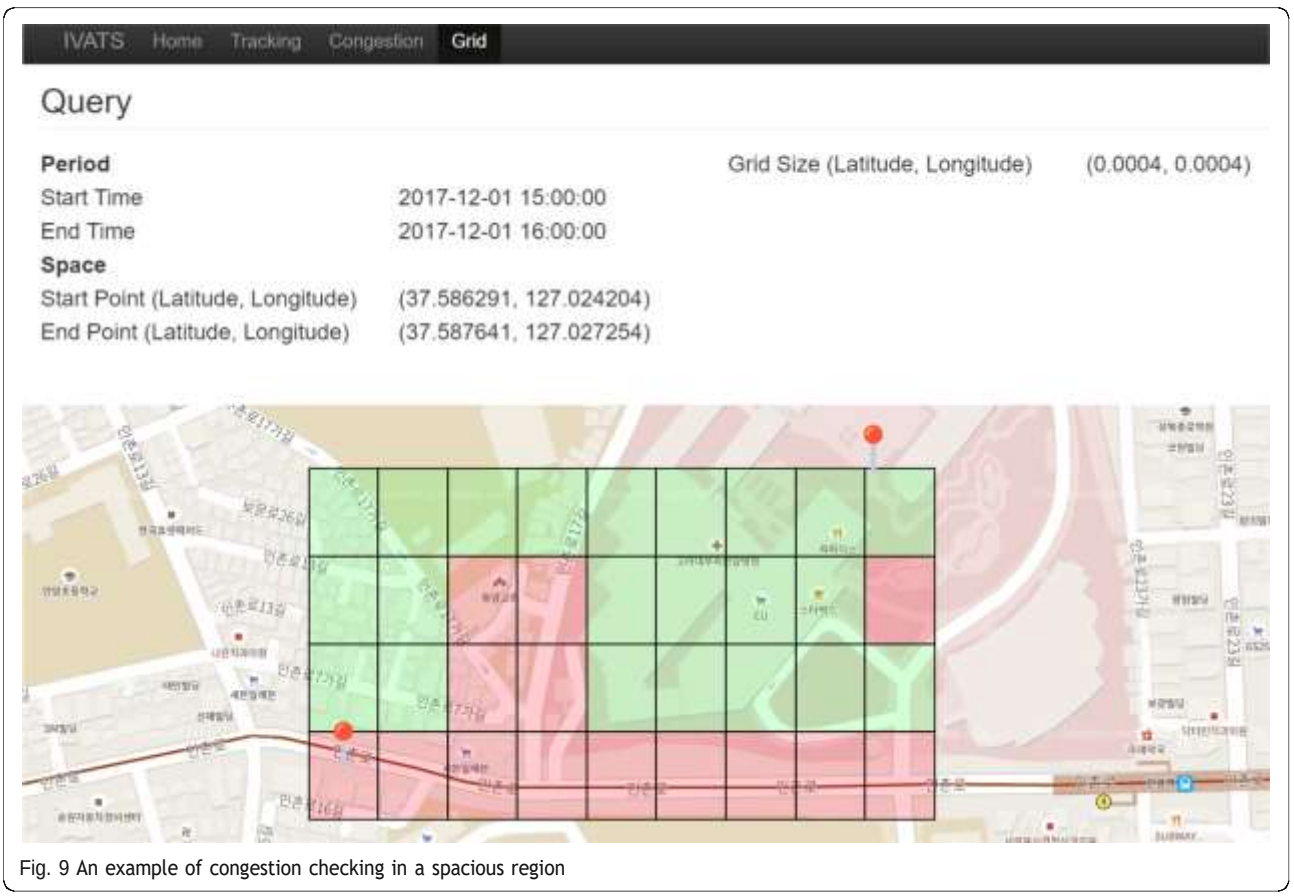

Fig. 9 An example of congestion checking in a spacious region

ISSN: 2278-4632
Vol-12 Issue-07 No.03 July 2022

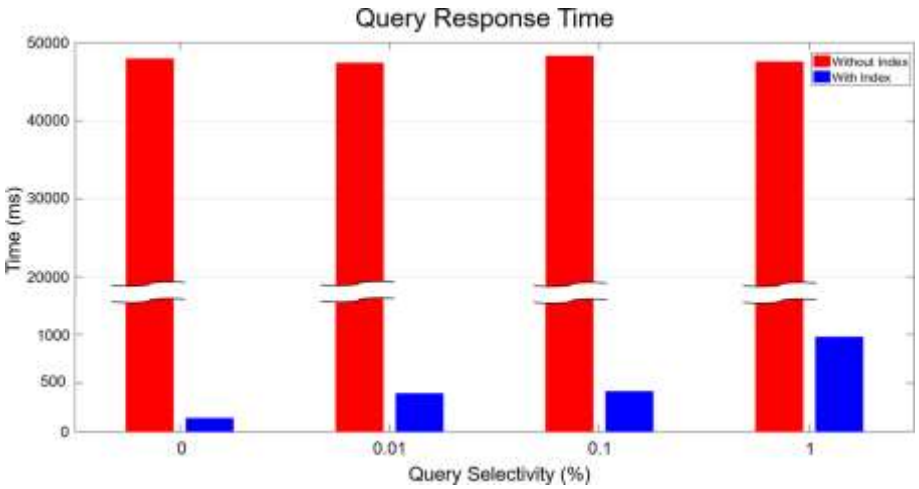

Fig. 10 Query response time depending on query selectivity

This type of query can be used to calculate the traffic congestion in a more spacious region at a specific time. Figure 9 shows such an example. Our system divides the given query area into grids and calculates the traffic congestion for each grid. The top part of the figure shows the query condition. In addition, the bottom part indicates how much the region in the grid is congested at the given time by using different colors. The green grid means that the number of the captured records in this region is under the average of the number of those in the query range, while the red grid means the opposite. This function can be used to find out a faster way to a destination or plan the construction of a new road.

4.3 Index efficiency

In the experiment, we evaluate the effect of our index structure by comparing the query response time when indexing is both used and not used for a user query whose range contains a portion of the entire data. We repeated this ten times. Figure 10 shows the query response times according to the query selectivity. The response times in the figure are the average of all response times for the queries. The query selectivity indicates the ratio of the query result to the total data. In the experiment, the query selectivity was set to 1%, 0.1%, 0.01%, and 0% (no relevant data). It can be seen in the figure that the response time when using an index structure differs depending on the query selectivity. On the contrary, the response times when not using an index structure are approximately identical regardless of the query selectivity. Overall, query processing can be achieved faster when using indexing. The difference in the response time between the two systems was the greatest when no data were in the query range, at which point the response time was approximately 300 times faster.

5 Conclusions

In this study, we proposed an integrated vehicle tracking system, IVATS, based on Kafka and HBase. Our system could assign a significant number of frames from diverse video sources, such as CCTV and car dashboard cameras, to processing nodes using Apache Kafka. Primary vehicle features such as plate number, time, and location data were extracted accurately from the frames using image and metadata processing. The feature data were stored in HBase clusters and retrieved for query processing. For effective query processing, we proposed an indexing structure based on R-Tree.

In the experiments, we demonstrated that our system can handle diverse user queries, including car tracking and traffic congestion, efficiently. Based on the data distribution, storage structure, Rowkey design, and indexing structure, our system can effectively handle real-time requirements of car tracking applications.

ISSN: 2278-4632
Vol-12 Issue-07 No.03 July 2022

References

1. D. Kim, E. Hwang, S. Rho, Multi-camera-based security log management scheme for smart surveillance. Secur Commun Netw 7, 1517 (2014)
2. CO Manlises, JM Martinez, JL Belenzo, CK Perez, and MKTA Postrero, Real-time integrated CCTV using face and pedestrian detection image processing algorithm for automatic traffic light transitions, in *Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*, 2015 *International Conference on* (IEEE, 2015), pp. 1
3. MML Elahi, R Yasir, MA Syrus, MSZ Nine, I Hossain, and N Ahmed, Computer vision based road traffic accident and anomaly detection in the context of Bangladesh, in *Informatics, Electronics & Vision (ICIEV)*, 2014 *International Conference on* (IEEE, 2014), pp. 1
4. Y. Guan, X. Wei, C.T. Li, Y. Keller, in *International Workshop on Biometric Authentication*. People identification and tracking through fusion of facial and gait features (Springer, Cham, 2014), p. 209
5. K. Muhammad, J. Ahmad, I. Mehmood, S. Rho, S.W. Baik, Convolutional neural networks based fire detection in surveillance videos. IEEE Access 6, 18174 (2018)
6. Apache Kafka. https://kafka.apache.org/. Accessed 30 July 2018
7. Apache HBase. https://hbase.apache.org/. Accessed 30 July 2018
8. Y. Nam, Y.C. Nam, Vehicle classification based on images from visible light and thermal cameras. EURASIP J Image Video Process 2018(5) (2018)
9. A Suryatali and V Dharmadhikari, Computer vision based vehicle detection for toll collection system using embedded Linux, in *Circuit, Power and Computing Technologies (ICCPCT)*, 2015 *International Conference on* (IEEE, 2015), pp. 1. https://doi.org/10.18063/cse.v0i0.405
10. J. Solanki, V. Rajguru, A. Saxena, Recognition of vehicle number plate using image processing technique. Control Syst Eng (2018)
11. J. Tarigan, R. Diedan, Y. Suryana, Plate recognition using backpropagation neural network and genetic algorithm. Procedia Computer Sci 116, 365 (2017)
12. Y. Rao, Automatic vehicle recognition in multiple cameras for videosurveillance. Vis. Comput. 31, 271 (2015)
13. YL Chen, TS Chen, TW Huang, LC Yin, SY Wang, and TC Chiueh, Intelligenturban video surveillance system for automatic vehicle detection and tracking in clouds, in *Advanced Information Networking and Applications (AINA)*, 2013 *IEEE 27th International Conference on* (IEEE, 2013), pp. 814
14. Google Maps. https://cloud.google.com/maps-platform/. Accessed 30 July 2018
15. Apache Hadoop. https://hadoop.apache.org/. Accessed 30 July 2018
16. Apache Spark. https://spark.apache.org/. Accessed 30 July 2018
17. C Ryu, D Lee, M Jang, C Kim, and E Seo, Extensible video processing framework in apache hadoop, in *Cloud Computing Technology and Science(CloudCom)*, 2013 *IEEE 5th International Conference on* (IEEE, 2013), pp. 305
18. G. Zhang, W. Q, L. Huang, B. Chen, *The optimization of task assignments onHadoop platform for large-number image processing* (2015)
19. C Sweeney, L Liu, S Arietta, and J Lawrence, HIPI: a Hadoop image processing interface for image-based Mapreduce Tasks, Chris. University ofVirginia (2011)
20. A. Jacobs, The pathologies of big data. Commun. ACM 52, 36 (2009)
21. Y.K. Kim, Y. Kim, C.S. Jeong, RIDE: real-time massive image processing platform on distributed environment. EURASIP J Image Video Process 2018, 39 (2018)
22. Apache Cassandra. https://cassandra.apache.org/. Accessed 30 July 2018
23. MongoDB. https://www.mongodb.com/. Accessed 30 July 2018
24. F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, R.E. Gruber, Bigtable: a distributed storage system for structured data. ACM Trans Comput Syst (TOCS) 26(4) (2008)
25. A Guttman, *R-trees: A dynamic index structure for spatial searching* (ACM,New York, 1984), Vol. 14, 2
26. H. Wang, A. Belhassena, Parallel trajectory search based on distributedindex. Inf. Sci. 388, 62 (2017)
27. N Du, J Zhan, M Zhao, D Xiao, and Y Xie, Spatio-temporal data index modelof moving objects on fixed networks using hbase, in *Computational Intelligence & Communication Technology (CICT)*, 2015 *IEEE International Conference on* (IEEE, 2015), pp. 247
28. H Sagan, Hilbert's space-filling curve, in space-filling curves (Springer, 1994), pp. 9
29. R.A. Finkel, J.L. Bentley, Quad trees a data structure for retrieval oncomposite keys. Acta Informatica 4(1) (1974)
30. X Chen, C Zhang, B Ge, and W Xiao, Spatio-temporal queries in HBase, in *Big Data (Big Data), 2015 IEEE International Conference on* (IEEE, 2015), Pp. 1929
31. X Xie, Z Xiong, G Zhou, and G Cai, WIT Trans Inf Commun Technol 49, 691 (2014)
32. B.W. Chen, W. Ji, F. Jiang, S. Rho, QoE-enabled big video streaming for large-scale heterogeneous clients and networks in smart cities. IEEE Access 4, 97 (2016)
33. N. Otsu, A threshold selection method from gray-level histograms. IEEE Trans Syst Man Cybern 9, 62 (1979)
34. R Smith, An overview of the Tesseract OCR engine, in *Document Analysis and Recognition, 2007. ICDAR* 2007. *Ninth International Conference on* (IEEE, 2007), pp. 629
35. N Beckmann, HP Kriegel, R Schneider, and B Seeger, The R*-tree: an efficient and robust access method for points and rectangles, in *Acm Sigmod Record* (ACM, New York, 1990), pp. 322
36. I. Kamel, C. Faloutsos, in *Proceedings of the 20th International Conference on Very Large Data Bases*. Hilbert R-tree: an improved R-tree using fractals (Morgan Kaufmann Publishers Inc, San Francisco, 1994), p. 500
37. Q. Zhu, J. Gong, Y. Zhang, An efficient 3D R-tree spatial index method for virtual geographic environments. ISPRS J. Photogramm. Remote Sens. 62, 217 (2007)
38. C. Faloutsos, S. Roseman, in *Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. Fractals for secondary key retrieval (ACM, New York, 1989), p. 247
39. B. Moon, H.V. Jagadish, C. Faloutsos, J.H. Saltz, Analysis of the clustering properties of the Hilbert space-filling curve. IEEE Trans. Knowl. Data Eng. 13, 124 (2001)
40. Y.K. Kim, C.S. Jeong, in *Proceedings of the 6th AIRCC International Conference on Parallel, Distributed Computing Technologies and Applications (PDCTA)*. Large scale image processing in real-time environments with Kafka (2017), p. 207