A Big Data Platform for Real-Time Video Surveillance

Thi-Thu-Trang Do
Faculty of Information Technology
Hung Yen University of Technology and Education
Hung Yen, Vietnam
trangdtt@utehy.edu.vn

Quang-Thinh Dam
Faculty of Information Technology
Hung Yen University of Technology and Education
Hung Yen, Vietnam
thinhquangshin@gmail.com

Tai-Huy Ha
Faculty of Information Technology
Hung Yen University of Technology and Education
Hung Yen, Vietnam
hataihuy2001@gmail.com

Quyet-Thang Huynh
School of Information and Communication Technology
Hanoi University of Science and Technology
Hung Yen, Vietnam
thanghq@soict.hust.edu.vn

Kyungbaek Kim
Department of Artificial Intelligence Convergence
Chonnam National University
Gwangju, Korea
kyungbaekkim@jnu.ac.kr

Van-Quyet Nguyen*
Faculty of Information Technology
Hung Yen University of Technology and Education
Hung Yen, Vietnam
quyetict@utehy.edu.vn

***Abstract*—Nowadays, smart house facilities are strongly developed with the support of multiple security cameras to protect not only a house but also a building. A large amount of video data is produced by these cameras every day. Therefore, traditional data management systems face challenges in collecting, storing, and analyzing big video data. In such systems, it is difficult to find objects and their actions from video surveillance in the building because of either the consuming time or the lack of intelligent technology support. In this paper, we propose a novel big data platform for real-time video surveillance analysis based on the combination of distributed data frameworks and intelligent video processing libraries. The proposed platform is able to collect both real-time video streams and historical video data by using Kafka and Spark Structured Streaming frameworks. Furthermore, the proposed platform provides an intelligent video processing module for object detection by using OpenCV, YOLO, and Keras libraries. To evaluate the proposal, we deploy the proposed big data platform and implement a web interface to support end-user to analyze video surveillance. Through the results of the initial video querying implementation, we show the viability of the proposed platform.**

***Index Terms*—Spark Structured Streaming, Kafka, Video Querying, Video Streaming, Video Surveillance**

I. Introduction

Recently, the volume of video data has increased dramatically on the internet from various sources such as Youtube, Facebook, and Tiktok. These unstructured video data are reservoirs of knowledge and have a direct relation to real-world events. It provides information about people's interactions and behaviors. Moreover, real-time video streams can help in behavior analysis whether it is of traffic or human patterns. The development of technology has also led to the development of security and healthcare systems. A large amount of video surveillance data is stored so that it can be processed when any event occurs. However, manually analyzing video surveillance will take a lot of time and effort. Therefore, video analysis platforms are researched and developed to manage and analyze these video data. These usually studied to evaluate and optimize data transmission throughput and speed. Bunrong Leang et al. [1] proposed a Hadoop ecosystem for supporting to several features in the manufacturing industry. Because the author only use Apache Hadoop and Kafka, ecosystem is quite limited in speed. To address this issue, we take advantage of Spark - a fast and general engine for large-scale data processing. Ayae Ichinose et al. [2] proposed a Streaming Video Engine (SVE) for uploading and processing videos in a distributed manner. That is a framework for real-time video analysis on Spark from which draw some conclusions about the dependence of throughput with the number of brokers or topic partitions. Kai Yu et al. [3] introduced a Video parsing and evaluation platform using Spark Structured Streaming and YARN with Kafka to solve problems existing in ordinary video surveillance systems. However, this platform does not support for extracting objects or properties of objects.

Mark Hamilton et al. [4] proposed a distributed image processing library which integrates OpenCV with Spark and Cognitive Toolkit - a deep learning library. However, this study is also limited to process images and does not provide any video processing APIs. Lei Huang et al. [5] introduced a method based on convolutional neural networks for recognizing objects in traffic video data. The video data is stored and processed by applying Spark. Kut et al. [6] proposed solutions based on both Hadoop and Spark for detecting edge using canny operator and line using Hough transform. Thus, most of the literature lacks the support for distributed feature extraction APIs.

To address this problem, a distributed video analysis framework is necessary. Md Azher Uddin et al. [7] proposed

* Corresponding author.

SIAT - a distributed video analysis framework for intelligent video surveillance. SIAT uses state-of-the-art distributed computing technologies to handle real-time video streams and batch video analytics to ensure scalability, efficiency, and fault tolerance. However, this framework does not have user interface. Melenli et al. [8] introduced a distributed image processing framework in real-time using Apache Kafka and Spark. They also used OpenCV and YOLOv3 to perform human detection and measured the distance between people. Besides, the authors have created an user interface which is used to define cameras, create notifications, or access real-time dashboards.

In this paper, we propose a novel big data platform for real-time video surveillance analysis based on the combination of distributed data frameworks and intelligent video processing libraries. The proposed platform be able to collect, analyze and store both real-time video streams and historical video data. We also implement a web interface to support end-user to analyze video surveillance and retrieve information from database.

II. Related Work

Real-time data processing and big data analytics have been attracting much attention recently [9, 10]. In particular, the development of the surveillance field has generated a huge of data daily leading to organizations must devise ways of handling this information since the existing techniques did not handle efficiently such a volume of data created at such a high rate. For example, Syafrudin et al. [11] introduced a real-time monitoring system that utilizes IoT-based sensors, big data processing, and a hybrid prediction model for the automotive industry. A Hadoop ecosystem is proposed to support several features in the manufacturing industry [1]. Zhang et al. [12] also proposed a cloud-based architecture that used Apache Kafka and Storm for real-time processing and Hadoop based MapReduce framework for batch video data processing. However, the Hadoop ecosystem and Kafka have a drawback which is the limitation of speed. It is the reason why Spark is studied and developed [13, 14].

Zhou et al. [15] introduced a distributed architecture, which can measure and monitor online Internet traffic. Ayae Ichinose et al. [2] proposed a Streaming Video Engine (SVE) - a framework for real-time video analysis on Spark, while Kai Yu et al. [3] introduced a video parsing and evaluation platform to solve problems existing in ordinary video surveillance systems. On the other hand, extracting objects or their properties of them are also attracted much attention and research.

BigDL [16] - a distributed deep learning framework using Spark is recently introduced, which allows users to develop deep learning applications. In this reference, they showed efficiency with object detection and image feature extraction. In addition, BigDL supports immensely efficient and scalable distributed training. Mark Hamilton et al. [4] also proposed a distributed image processing library that integrates OpenCV with Spark and Cognitive Toolkit. However, they do not provide support for video processing. In contrast, Huang et al. [5] proposed a method based on convolutional neural networks for recognizing objects in traffic video data and using Spark to store and process this information. However, most of the existing proposals do not yet support the retrieval of information from the analyzed and stored data.

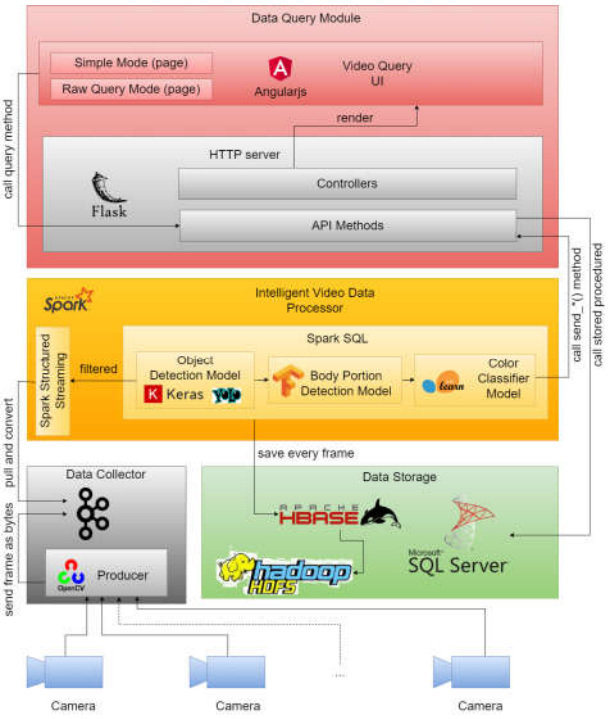

Fig. 1. The system architecture.

Kut et al. [6] proposed solutions based on both Hadoop and Spark for detecting edge using the canny operator and line using Hough transform, while SIAT [7] is proposed by Md Azher Uddin is a distributed video analysis framework for intelligent video surveillance. However, they do not implement a user interface. Melenli et al. [8] introduced a distributed image processing framework in real-time using Apache Kafka and Spark, which can detect humans, measure the distance between people and implement a user interface.

III. Design of a Big Data Platform for Video Surveillance

A. Architecture of the Proposed Platform

We propose a platform for collecting and analyzing multiple video data in real-time with four components including Data Collector, Data Storage, Intelligent Video Data Processor, and Data Query Module. Figure 1 presents the architecture of our proposed big data platform. The first component is the Data Collector which contains two sub-components: Kafka cluster and producers. Brokers in the Kafka cluster are managed by Zookeeper. Producers have the responsibility for extracting data from real-time videos and then sending them to the brokers to store. The collected data from video by the Data Collector will be consumed by the third component - the Intelligent Video Data Processor which has a processing mission using Spark Structured Streaming and SparkSQL. Raw data after being consumed by the processing module is contained by the second module - The Data Storage. The processed data by the Intelligent Video Data Processor will be sent to the fourth component - the Data Query Module via the HTTP server. Then, the HTTP server will send back the data to the Data Storage – SQL Server to store for serving the Data Query module. We just save several frames in SQL Server to

reduce the vast number of frames, instead, we save it into Hbase - a distributed NoSQL database and it saves the data as a file in Hadoop Distributed File System (HDFS), so the system can serve query tasks more rapidly. The web client in the Data Query module is a user interface that supports humans to query data from the SQL server.

B. Design of Real-Time Data Collector Module

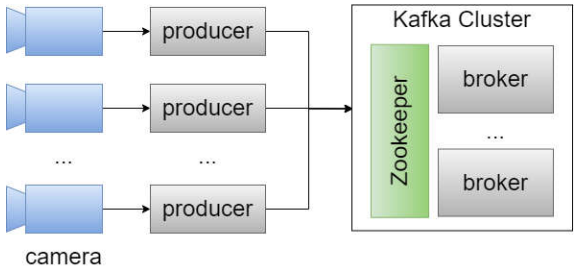

Fig. 2. The Kafka system.

Figure 2 shows our design for collecting data using Kafka. This module is designed to collect the generated data from cameras. We do not want any data loss when receiving data from producer, so we design this component according to replicas and use Zookeeper to manage nodes in the cluster. We all know that the generated data from cameras are videos that are huge collections of images. These images move fast enough for us to see the things call animations. Producers have the responsibility for collecting these images and then converting them to three-dimension arrays called frames by a powerful computer vision library – OpenCV. The problem is that a large number of frames are generated every second. Therefore, instead of sending all of them, we just send one per two seconds to ensure that we have enough bandwidth to transfer and the brokers have enough storage to store them. However, we have a second problem: the matrices can not be transferred by the producers. So, we convert them to bytes strings to send more easily.

C. Design Choice of Data Storage Module

We design two sub-components in the Data Storage module. The first component is Distributed NoSQL database named Hbase. We use this database because it is fast when reading, writing, or random data. Furthermore, it can store massive data because it bases on the Hadoop Distributed File System (HDFS) which stores data safely and distributedly. The second component is SQL Server, we use it to store the processed data by the Intelligent Video Data Processor module.

1) Design of Basic Distributed Storage: We need basic storage for storing raw frame data at every time. These data are filtered by the Intelligent Video Data processor module. These raw data are usually sequentially contained in text files. In this case, our type of data is byte-string which is a fixed-length array of bytes. A byte is an exact integer between 0 and 255 inclusive. These raw data are the basic data for processing and we use Hbase as basic storage for them.

2) Design of Storage for Querying Data: In our platform, we also consider an easy API for querying data with SQL language. For this, we use SQL Server because it is simple to implement and easy to integrate into our platform. Our platform is designed to trace objects and extract properties from them, so we have to design a schema that is suitable for this case. We have six tables to contain information extracted by the Intelligent Video Data Processor module.

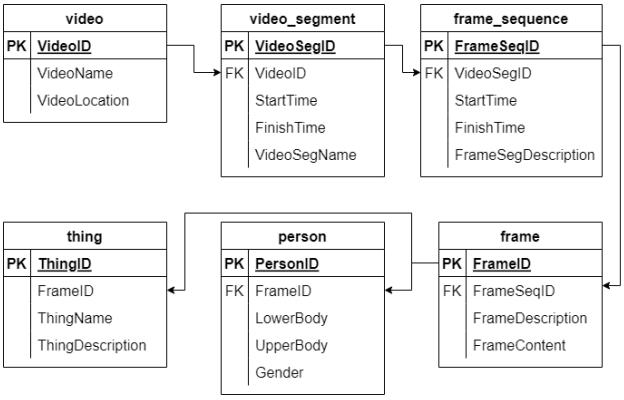

Fig. 3. Database schema.

- **Video**. It contains information about cameras which were installed in our system. Each camera has four properties particularly: ”VideoId” - a unique key to distinguish from others, ”VideoName”- the name of the camera, ”VideoLocation” - the name of the location where the camera was installed.
- **Video_segment**. It contains information about segments of each video. Per ten minutes, we create a new segment. The table has five properties: VideoSegId - a unique key to distinguish from others, VideoId - the camera’s key producing this segment, StartTime and EndTime are times when video starts and stops, VideoSegName - the name of the video.
- **Frame_sequence**. This table contains information about the period when the processing module detected an object in a frame. For each object detected by the processing module, we create a new frame_sequence record.
- **Frame**. The fourth table is Frame which contains frames from cameras that have objects detected by the processing module. We have to encode the frames to base64 to store them in the database more easily.
- **Person** and **Thing**. Person and thing contain information about objects detected by the processing module. Table ”person” contains information about the properties of humans and table ”thing” contains information about anything that is not humans. The Fig.3 illustrates a schema of the database.

D. Design Choice of Intelligent Video Data Processor Module

In order to consume the data stream from the Kafka cluster, we design this module including AI models. The AI models have the responsibility for detecting objects that appear in frames sent by Kafka, then extracting properties from them. We use a built-in library – request to send output data to the HTTP. Leveraging the real-time querying power of SparkSQL, we built a user-defined function (UDF) and used models inside server.

1) Design Choice of Image Processing models: Figure 4 demonstrates the processing sequence of the Spark Structured Streaming program. The input of the Intelligent Video Data

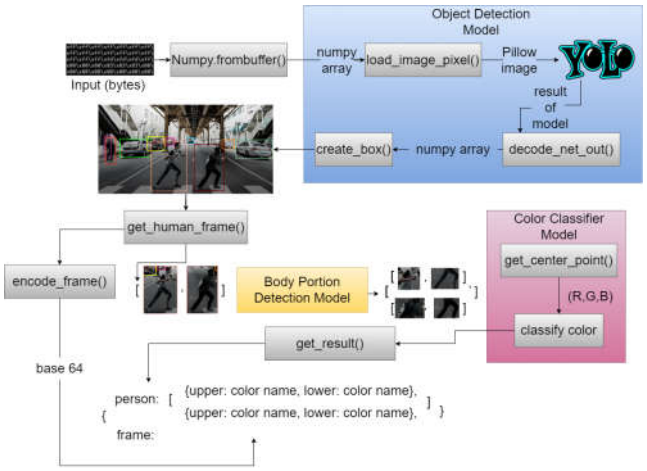

Fig. 4. Processing progress.

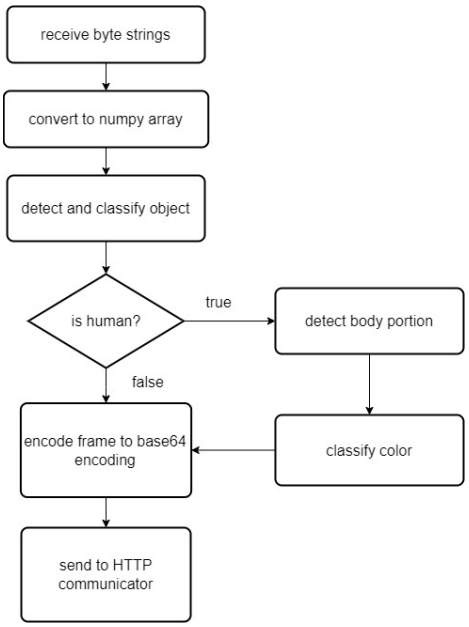

Fig. 5. Processing logic.

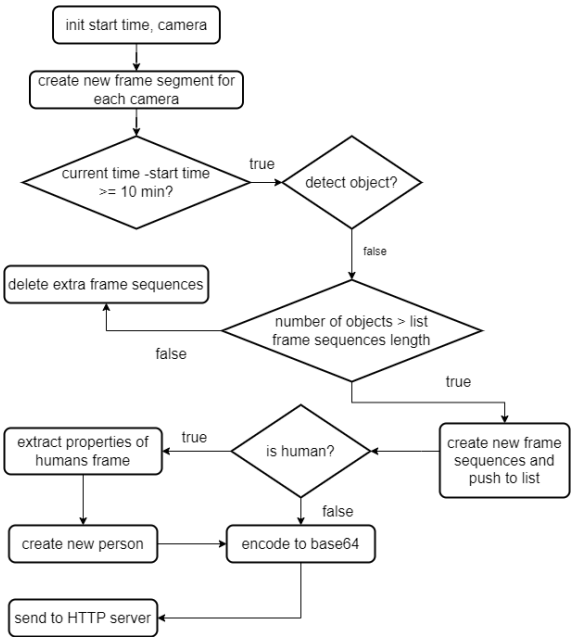

Fig. 6. Database business.

processor module is a collection of byte strings filtered by SparkSQL after receiving data from Kafka clusters. So, we use a powerful computer vision library – OpenCV to convert them to frame type (NumPy array) because AI models can not calculate byte string data. In order to detect objects in the converted frame, we use a pre-train AI model – YOLOv3. But, there is a problem that YOLO can not integrate with Spark Structured Streaming. Because it is developed base on the DarkNet platform which creates a new thread whenever it runs. However, Spark Structured Streaming does not accept any thread except its, therefore, we have to convert the DarkNet model to another type of model that can be excepted by Spark Structured Streaming. So, we use Keras to convert the YOLOv3 model to the Keras model. We then use body portion detection to detect the upper body portion and lower body portion of human images classified. To train this model, we use a pre-train model called "Tflite Model Maker" which is developed by Google with 20000 labeled images by Labelimg tools. To collect images for training the model, we downloaded some videos recorded by security cameras from Youtube which were public. Then we use the Yolov3 model to get a human object displayed in each frame in each video and save them to a folder. After that, we used the LabelImg tools to label the images. In each human frame, start point from the shoulder to the end of the back, we labeled this part as "upper" and from the end of the back to the ankle we labeled it as "lower". For the train set, we use 10000 images, 5000 for the test set, and 5000 for validating set. Then, we use the Tflite model maker to train our custom model with the labeled dataset. The model after training has an accuracy is 0.81. Finally, we use a simple machine learning model to classify colors in body portion images. It is implemented by using the KNN algorithm built in Scikit-learn. The output of the whole processing process and related information (object properties) are sent to the HTTP server. Especially, the out frames are converted to base64 encoding to be easy to store in the database. Figure 5 shows the image process and information extraction process.

2) Design of Database Business: The Fig.6 shows the design of the database business. As we design the schema of the database, we need to design a business that is suitable for storing data in the database. First, we initialize the start time, and information about the cameras installed in our system. We then create new frame segments for each camera. Because, after 10 minutes, we need to create a new segment, we need to check the running time by subtracting the starting time from the current time. If the running time is greater than or equal to 10 minutes, new frame segments are created. Next, we will check the objects detected by the YOLO model. If the number of detected objects is greater than the number of frame sequences, new frame sequences will be created. The number of new sequences is equal to the number of extra objects. If the number of detected objects is smaller than the

number of sequences, the extra sequences will be removed. Then, for human frames, we extract properties from them. After that, we convert the frames to base64 including things' frames. Finally, we send this information to the HTTP server.

E. Design Choice of Data Query module

We need a web API to query data from the database and transfer data from the data processing module to the database. To do this, we choose the Flask library to create one because of its lightweight and simplicity. Our API has 6 HTTP methods as the following:

- **send_segments** method. It is used to send segment information from the data processing module to the HTTP server.
- **send_sequences** method. It is used to send frame sequence information to the server.
- **send_frames** method. The send_frames has the responsibility for transferring base64 frames from the processing module.
- **send_people** and **send_things** methods. The send_people method and send_things method are used to send information about things and humans' properties.
- **index** method. It is used to return an interface to the user.

These methods use PyOdbc - a built-in library to execute stored procedures written in the database. The query method with its parameter – command will send a command to the SQL server to execute the command and then receive the result.

IV. Implementation and Experiments

A. Environment Settings

We deployed the platform in a cluster including 5 machines, in which, two machines are used to collect data from two cameras and three other machines are to install Hadoop and Spark workers as a cluster. The cluster includes two workers and a master. The master machine has 4GB RAM and 4 CPUs and workers have 2GB RAM and 2 CPUs on each machine. We use another machine that has 4GB RAM and 4 CPUs to install the SQL server database and the HTTP server. And the 2 machines which are used to collect data, which we use to deploy our Kafka cluster, have the same configuration as the workers.

B. Implementation of Web User Interface for Querying Video Data

We designed a web client to support the users to query data from SQL Server more easily. To build this client, we used AngularJS and call the API we built. The client has two modes: Simple Mode and Raw Query Mode. In addition, to run the platform we had to add the cameras' information to the database first and configuration file. Then, we started the SQL server and HTTP server. After that, we started the Hadoop system and Spark Structured Streaming job with our workers. Finally, we started all cameras and the Kafka cluster.

1) Simple Mode: The user interface shown in Fig.7 has two sections. The first section is the input of searching. It is used to type the properties of an object and the information about place and time. The first component of the first section is a list of cameras that can choose to specify which camera the users want to query. The second component is "Video Time" advocating the users to query the start and end time of the video to limit the range of time in which the object appeared. The third component is used to query the properties of humans appearing in the camera. The last component supports the users to query things existing in the camera.

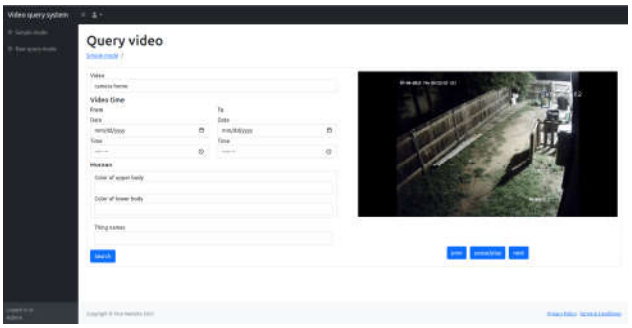

Fig. 7. The user interface of Simple Mode.

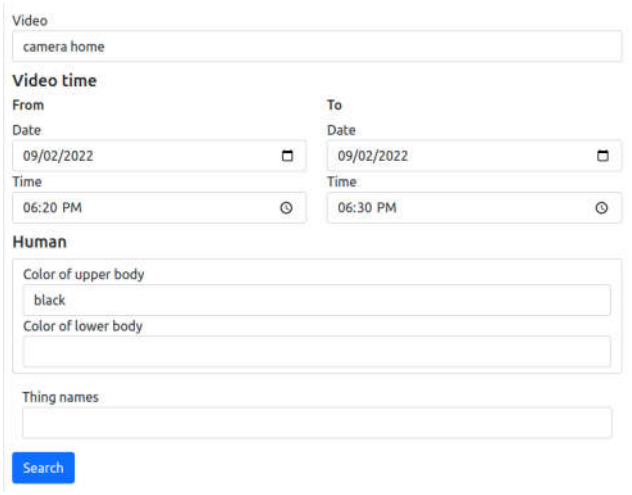

Fig. 8. Query the person with properties display in specified camera in range time.

The second section displays the result of the search as a list of images. We designed the second section with three buttons to move between images or auto-play as a video. For example, we queried the data from the camera named "home camera" from 18:20 to 18:30 at 2nd December to find the person who wore a black shirt. Every search inputs are shown in Fig.8 and the result of searching is demonstrated in Fig.9

In the other case, we searched a man who wore a grey shirt who appear in all cameras and the all time, the result showed in Fig.10.

2) Raw Query Mode: We built the raw query mode to enhance the query ability of the system. It has two sections, the first section displays the result as a list of images, and the second section is used to write SQL commands. The interface was designed as Fig.11. This mode can be used to query with a complex condition that the simple mode does not support. The example below demonstrated how to search with multiple cameras in multiple time ranges. To display the result as an image and display the time at which the frame is recorded, we need to join two tables: frame and frame_sequences. The script which we used to query, is shown in Listing 1 and the result of the query is illustrated in Fig.12.

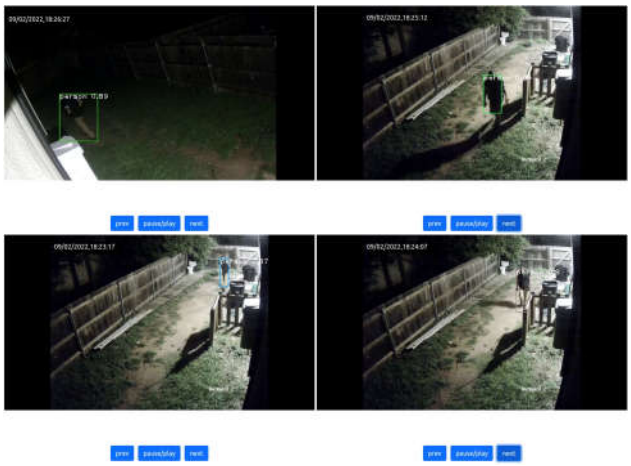

Fig. 9. The result of the searching in Fig. 8.

Fig. 10. The result of the searching.

```
SELECT * FROM frame f
JOIN frame_sequence fs
ON f.FrameSegID = fs.FrameSegID
JOIN video_segment vs
ON vs.VideoSegID = fs.VideoSegID
JOIN video v ON v.VideoID = vs.VideoID
WHERE
    VideoLocation = 'yard'
    OR VideoLocation = 'petrol station'
    AND (
        CONVERT(DATETIME, '2022-09-11 21:35:23.000')
     <= fs.StartTime
        AND fs.StartTime <= CONVERT(DATETIME, '
    2022-09-11 21:50:27.000')
    )
    OR (
        CONVERT(DATETIME, '2022-09-11 22:35:23.000')
     <= fs.StartTime
        AND fs.StartTime <= CONVERT(DATETIME, '
    2022-09-11 22:50:27.000')
    )
```

Listing 1. query images ordered by time

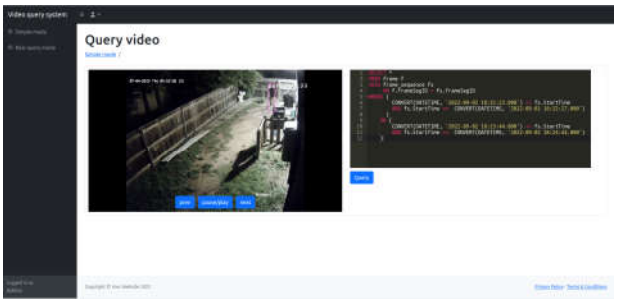

Fig. 11. User interface of Raw Query Mode.

Fig. 12. The result of query command in Listing 1.

V. Conclusions

This paper proposed a design and implementation of a big data platform for real-time video surveillance. The proposed platform was designed with the aim that supporting user to manage their cameras system more efficiently by using the strong analytic power of big data tools integrating with a set of AI models. The proposed platform is able to collect both real-time video streams and historical video data by using Kafka and Spark Structured Streaming frameworks. It also provides an intelligent video processing module for object detection by using OpenCV, YOLO, and Keras libraries. Through the results of the initial video querying implementation, we show the viability of the proposed platform. In the future, we will work on optimizing AI models and integrating the platform with distributed search engines instead of using the SQL language in the relational database to improve search speed.

Acknowledgements

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the Innovative Human Resource Development for Local Intellectualization support program (IITP-2022-RS-2022-00156287) supervised by the IITP (Institute for Information & communications Technology Planning & Evaluation).

References

[1] Bunrong Leang et al. “Improvement of Kafka streaming using partition and multi-threading in big data environment”. In: *Sensors* 19.1 (2019), p. 134.

[2] Ayae Ichinose et al. “A study of a video analysis framework using Kafka and spark streaming”. In: *2017 IEEE International Conference on Big Data (Big Data)*. 2. IEEE. 2017, pp. 2396–2401.

[3] Kai Yu et al. “A large-scale distributed video parsing and evaluation platform”. In: *Chinese Conference on Intelligent Visual Surveillance*. 3. Springer. 2016, pp. 37–43.

[4] Mark Hamilton et al. “Flexible and scalable deep learning with MMLSpark”. In: *International Conference on Predictive Applications and APIs*. 4.

[5] Lei Huang et al. “Enabling versatile analysis of large scale traffic video data with deep learning and HiveQL”. In: *2017 IEEE International Conference on Big Data (Big Data)*. 5. IEEE. 2017, pp. 1153–1162.

[6] Seda Kul et al. “Event-based microservices with Apache Kafka streams: A real-time vehicle detection system based on type, color, and speed attributes”. In: *IEEE Access* 9.6 (2021), pp. 83137–83148.

[7] Md Azher Uddin et al. “SIAT: A distributed video analytics framework for intelligent video surveillance”. In: *Symmetry* 11.7 (2019), p. 911.

[8] Sadettin Melenli and Aylin Topkaya. “Real-time maintaining of social distance in covid-19 environment using image processing and big data”. In: *The International Conference on Artificial Intelligence and Applied Mathematics in Engineering*. 8. Springer. 2020, pp. 578–589.

[9] Lucy Linder et al. “Big building data-a big data platform for smart buildings”. In: *Energy Procedia* 122.10 (2017), pp. 589–594.

[10] Tom Wilcox et al. “A Big Data platform for smart meter data analytics”. In: *Computers in Industry* 105.11 (2019), pp. 250–259.

[11] Muhammad Syafrudin et al. “Performance analysis of IoT-based sensor, big data processing, and machine learning model for real-time monitoring system in automotive manufacturing”. In: *Sensors* 18.12 (2018), p. 2946.

[12] Weishan Zhang et al. “A video cloud platform combing online and offline cloud computing technologies”. In: *Personal and Ubiquitous Computing* 19.13 (2015), pp. 1099–1110.

[13] Jian Fu, Junwei Sun, and Kaiyuan Wang. “Spark–a big data processing platform for machine learning”. In: *2016 International Conference on Industrial Informatics-Computing Technology, Intelligent Technology, Industrial Information Integration (ICIICII)*. 16. IEEE. 2016, pp. 48–51.

[14] R Shyam et al. “Apache spark a big data analytics platform for smart grid”. In: *Procedia Technology* 21.17 (2015), pp. 171–178.

[15] Baojun Zhou et al. “Online internet traffic monitoring system using spark streaming”. In: *Big Data Mining and Analytics* 1.14 (2018), pp. 47–56.

[16] Jason Jinquan Dai et al. “Bigdl: A distributed deep learning framework for big data”. In: *Proceedings of the ACM Symposium on Cloud Computing*. 15. 2019, pp. 50–60.