Multi-channel Video Streaming Technology for Processing
Based on Distributed Computing Platform

Wei Cheng¹,*, Yanyang Tang¹ and Wangdong Yang²

¹School of Marxism, Hunan University of Humanities, Science and Technology, Loudi, Hunan417000
²School of Marxism, Hunan University, Changsha, Hunan410000

*84148207@qq.com

Abstract. With the rapid development of the Internet, sensor network, and mobile Internet technology, a large number of data sets are continuously generated in the form of streaming in various application fields. At the same time, the processing of stream data has drawn more and more attention because of its application in different situations. To meet the urgent need for the processing of streaming data, there are many computing engines on the market, such as Spark and Flink. Traditional data-flow analysis has many problems in the process of streaming data, such as the high time delay, weak extensibility, and bad adaptability. In order to improve them, people use Kafka as the intermediate-cache and the computing engine that uses Spark Structured Streaming as the streaming data to realize the effective process of multichannel flow data. Considering the present demand for low latency and high flux that are analyzed by urban management in traffic video, processing the data of multichannel traffic video on distributed computing platform realized the tracking and search on motor vehicles, non-motor vehicles, and pedestrians of multiple roads and intersections.

Keywords: Spark Structured Streaming, Apache Spark, Distributed platform, Streaming data processing, Traffic video analysis
1. Introduction

With the rapid development of the Internet, sensor network, and mobile Internet technology, a large number of data sets are continuously generated in the form of streaming in various application fields. Due to its wide applications in various fields, such as analysis of real-time transactions, analysis of Internet of Things data, intelligent advertising, log processing, and analysis of measurement, convective data processing has drawn more and more attention. In order to meet the needs of applications, frameworks of streaming data processing have emerged, such as Apache Flink [1], Apache Spark [2], and Apache Storm [3]. Using these big data processing platforms to change the processing mode of traditional streaming data, such as video streaming, is conducive to making some applications better deal with the challenges brought by the era of big data.

In recent years, with the popularization of network technology, the original scattered video resources are being prompted towards the direction of unified and centralized processing. The increasingly intensive deployment of cameras in industries such as smart cities means that new video sources will increase at every moment. Changes of video resolutions from 360p, 720p to 1080p, and even of 4K show that video information is getting larger and larger. These increasing video data not only brings the demand for video storage but also poses a challenge to the video processing technology [4]. Large-scale data, diverse data sources, and application requirements for video timeliness demand efficient technology for processing a large amount of multi-channel videos.

A large amount of video data needs to be processed in real-time to provide large-scale and real-time video analysis solutions. The following challenges are faced:

1. How to select a feasible systematical design to realize a feasible computing architecture oriented to the multi-channel video stream to meet the real-time requirements, while taking both the huge video stream and network communication cost of large-scale control terminals into account;
2. How to build a platform with good scalability and flexibility;
3. How to make full use of distributed resources. In this work, we put forward a feasible multi-channel video streaming technology for processing based on a distributed computing platform by using Spark structured flow and Kafka. And by dealing with traffic video streams in the field of urban management, we can make the tracking and search on motor vehicles, non-motor vehicles, and pedestrians of multiple roads and intersections. Moreover, we can obtain real-time traffic flow and forecast the congestion situation of these intersections to realize urban automation management further.

2. Related work

Many kinds of research have been conducted on multi-channel video stream processing [5-7].

First, cloud computing can traditionally provide low-cost and flexible computing resources with almost unlimited capacity, so some traditional focuses are to use cloud computing to solve the problem
of multi-channel video processing. However, the video stream is basically used as the basic scheduling unit, and video processing capacity and physical resources are highly coupled [5]. Because every task of video stream processing will result in huge resource consumption of a server, the traditional scheduling scheme of the coarse-grained task will lead to a lot of resource fragmentation and underutilization. The video analysis system based on the Hadoop computing framework used by Zhao et al. [6] first stores the video to HDFS and then uses MapReduce for distributed processing. However, based on file system processing, Hadoop results in a large number of IO operations and high delay. Apache Spark has the advantages of Hadoop and MapReduce, but unlike MapReduce, the intermediate output of it can be kept in storage, which makes it no need to read or write HDFS, thus the operation on disk IO is greatly reduced. Zheng Jian [7] et al. proposed that using Spark Streaming[8] and Kafka cache as a solution to process video streaming. However, Spark Streaming uses the microbatch processing mode, where the data flow is divided into microbatches at a specified time interval and then the data is batched for processing, mainly for bounded batch data processing. But a real flow should be continuous and have no boundaries [9]. Spark Structured Streaming[10], which eliminates the concept of the batch, can truly support continuous stream processing and can be processed immediately as soon as data items arrive. Compared with other Streaming computing frameworks, its outstanding advantage is that it can realize fault-tolerant recovery and seamless integration of data analysis, machine learning, and other tools in an efficient way.

In this paper, we mainly use Spark Structured Streaming and Kafka message-oriented middleware to process multi-channel traffic video and conduct real-time detections of vehicle flow and a pedestrian on traffic roads, so as to provide a reliable basis for traffic management, improve the work efficiency of traffic management and maintain good traffic order. At the same time, the platform has obvious advantages in extensibility and fault tolerance.

3. System Architecture

For high flux streaming data, clustering is the preferred way to ensure it. For data segmentation, balanced task allocation, cluster communication, cluster resource management, and other functions, distributed big data processing frameworks, such as Spark, Hadoop, and Flink can be used. They provide a general distributed parallel computing platform for data processing and greatly reduce the complexity of system development [11]. It can also directly increase the computing resources to improve the processing performance and computing scale of videos. Calculating based on storage, the Spark is faster than the Hadoop that calculates based on the file system and can directly use Spark Standalone mode to manage the cluster in a unified way, reducing the difficulty of the deployment of the Spark in the cluster, making it easier to use to add new computing resources, supporting the Spark Structured Streaming in using advanced DAG scheduler, the query optimizer and physical execution engine and also realizing the high performance of batch and flow data processing. Spark also
supports batch processing, interactive queries, machine learning, and graph computing, and has a complete ecosystem that can significantly reduce system update and maintenance costs. So we use Spark as the basic resource management and task execution framework. The architecture of the whole system is shown in Fig.1.

The whole system mainly consists of two parts: the Collector and the Processor. The Collector of flow data receives the real-time video stream from the camera, uses OpenCV to parse video streams, converts it into continuous video frames, and stores video frames in the Stream Data Buffer based on Kafka distributed message queues cluster construction. The Processor of the data stream takes out a data frame to use OpenCV and other library functions for target recognition, license plate recognition, object tracking, etc. Finally, store the metadata information analyzed from the video stream into the storage.

Among them, Stream Data Buffer can decouple data collection and data processing, thus solving the problem of mismatch of processing rate between Data Stream producers and Data Stream users. Lightweight distributed Kafka message queue is selected as stream data cache in memory, and Zookeeper is used to coordinate the Kafka cluster, so as to realize load balance and fault recovery of the Kafka cluster. Collectors of the data flow will store data collected by different cameras into different partitions of the same Topic.

The Processor of data flow is composed of Spark as a resource management cluster, which is divided into Master Node as task allocation and resource coordination Node and Worker Node task execution Node. The Master will allocate different partitions of the same topic according to the use of each Worker's usage of resources, and then each Worker will continuously read data from the specified partition. As long as the data reach the Kafka, Spark Structured Streaming immediately reads data, and then the user Trigger (Trigger) reads the interval of the data stream into a chart, which is to convert the data stream data item in the Tab of a line. Next, use SparkSQL execution engine Stream Execution to deal with each line of data flow analysis and optimization, and the results will be updated to the DataStorage for storage.
4. Traffic video stream processing

In the development of the smart city, the increase of camera deployment can timely analyze data such as detection and tracking of motor vehicles, non-motor vehicles, and pedestrians and traffic congestion from multi-channel video stream data, which plays an important role in urban management decision. However, the large traffic flow, multiple sources, and high timeliness requirements of traffic video data pose great challenges to the load balancing, scalability, and data cache of the data processing system. Therefore, we analyze and process traffic video data for this problem. The specific process is as follows.

4.1 Stream data acquisition

Traffic video data collection is carried out by each server to get data from the designated IP cameras. Then the video is parsed and is stored in the Kafka message queue. Apache Kafka provides high-throughput data processing capability and supports the process of stateful data flow and low-latency distributed queries. However, the Kafka message queue has certain performance advantages in processing text-type data. So we must convert continuous video data into a series of video frames and then encapsulate it into JSON format. The specific data format is shown in Fig.2.

![Data encapsulation format](image)

Since different cameras may correspond to different specifications, such as encoding method, resolution, number of frames per second, and position information generated by frames, we must identify the source of each frame data for further analysis. So we introduce CameraID as the identifier for the different cameras.

TimeStamp is the time when the video collector processes the video frame. In the process of sending the video to Kafka, there may be a delay. TimeStamp is used to ensure that the video processor can restore the original video stream data when processing the video.

FrameData is the information of video frames. We use VideoCapture in the OpenCV video processing library to decode the video stream and convert it into a continuous Mat frame structure, which is the basic frame structure in OpenCV. In order to simplify Kafka's storage and data acquisition, the data resolution (e.g. 640x480) was pre-adjusted uniformly. FrameData is the base64-encoded string that is converted from the Mat byte array for that frame after adjusting sizes. Since the Mat needs to keep the details of the frames (that is, rows, columns, and types) constant, then
it can be converted to serializable (byte array) form. So we also need to encapsulate the rows and columns that correspond to Row, Cols, and image Type information of the video frame respectively. Finally, we convert the encapsulated video frame data into JSON format and then send the JSON message to the Kafka server by Kafka client. The Kafka client sends videos with different video Ids to different partitions in the same theme.

4.2 Stream data processing

At present, the mainstream distributed streaming data patterns is mainly divided into two categories: one is the microbatch processing system that divides streaming data into tiny batches for processing, and the other is the streamlined flow data processing. Both models have effective fault tolerance, one-time semantics, and analysis of batch and stream data[11].

In the streamlined flow data processing model, the next batch processing can be carried out only when the processed data items are ready without forming the whole batch processing. Therefore, the model implements low latency. Apache Flink implements a pipelined flow processing model to support a true native flow processing engine. But this also means that it is difficult to achieve fault tolerance without affecting throughput because for each record we need to trace and checkpoints after processing. But state management is easy because long-running processes can easily maintain the required state.

In the batch model, time intervals that are predefined split the input stream into microbatches. Spark Streaming and Spark Structured Streaming[9] process the flow data in this way. They split the stream data into distributed data sets (RDD), which allows users who use the same API to switch between batches and streams. Because this model has to cache a batch of data before a batch operation can take place, the latency of the task is high. But it is convenient to provide fault tolerance because it is essentially a batch process and throughput is high and because the processing and checking points will be done in one set of records. However, this will cost some delay and efficient state management will be a maintenance challenge.

Flow processing is a powerful tool, and all kinds of flow computing platforms have their own advantages. However, many streaming system platforms such as Flink which needs to consider a variety of complex API semantics. For example, the user needs to specify window mode, trigger mode, and trigger refinement mode for each operation. Considering engineering implementations, we prefer the expressions that are easy to use, operate, and integrate into an application's Spark Structured Streaming using the incremental query model to simplify an application. And the main aim is to support end-to-end real-time applications that integrate flows with batch processing and interaction analysis. Moreover, the structured code generation engine circulating through Spark SQL achieves high performance and can exceed Apache Flink by up to 2 times [12].
In the traffic video stream processing environment, it is mainly responsible for taking the encapsulated frame data out of the memory Kafka to track vehicles and pedestrians and predict congestion. The specific data flow diagram is shown below.

![Data flow diagram of video analysis](image)

The whole video stream processor is mainly divided into three steps:

First, Spark Structured Streaming connects with Kafka server. Each Worker reads a set of data frames in the corresponding partition in parallel, orders them according to time, and then parses each frame information.

Second, frame bytes are added to the adjusted YOLO model for target recognition, so as to identify motor vehicles, non-motor vehicles and pedestrians marked on the original frame. Meanwhile, identified motor vehicles are clipped and added to the HyperLRP library for specific license plate detection and recognition.

Third, the data is encapsulated, and the specific data format is shown in the Fig. At last, filter out the identified result FrameResolution and the results that are the same as CamerID and licencePlateList and continuous in TimeStamp and then convert them into JSON format and save in Kafka.
4.3 Results of streaming data analysis

We directly send the processed video frames to the user terminal for display, and the processing effect is shown in the Fig. If a user queries a vehicle, the data can be searched from Kafka and sorted according to time. And merging the data of the same ID in a continuous time period can find out the camera ID of the vehicle in different time periods. Finally, the location of the vehicle can be tracked according to the camera ID. The specific data flow diagram is shown in the Fig.

5. Summary

Real-time data flow analysis of multi-channel video massive data streams requires high capacity and throughput and data processing engines. We built a collector component of a video stream by using OpenCV and Kafka and it can receive video streams from different sources and send them to the stream data buffer component. Kafka, as the stream data buffer component, provides persistent storage of stream data. The processor component of the video stream is developed by using OpenCV and
Spark Structured Streaming, which can effectively process multi-channel traffic video streams. When processing video streams, we found that different applications have different needs in delay and performance. For example, the streaming video display can receive a certain delay as a buffer and have no strict requirement of time delay. But when processing warning applications of video analysis, we found that it has higher requirements on time. So we will make different programming models for different needs of different applications in the next step.

Acknowledgments
This paper is a national key research and development program, Item no: 2018YFB1003401. Research and Practice of Computer Network Simulation Teaching Reform in teaching Reform Project of Hunan University of Humanities and Science and Technology (RKJGY1401), and Information System Research of Youth Topic Mashup Technology of Hunan University of Humanities and Science and Technology (2105QN13)

References
[1] Apache Storm Project. Available Online: http://storm.apache.org/
[2] Apache Flink Project. Available Online:http://flink.apache.org/
[3] Apache Spark Project. Available Online: http://spark.apache.org/
[4] Deng Y Application and Challenge of Internet of Things in smart Cities J. China Public Security 3 pp158-161
[5] Yang X , Zhang H , Ma H and et al 2016 Multi-resource Allocation for Virtual Machine Placement in Video Surveillance Cloud C. International Conference on Human Centered Computing. Springer International Publishing
[6] X Z, H M, H Z, Y T, and Y K 2015 Hvpi: extending hadoop to support video analytic applications Proc. IEEE International Conference on Cloud Computing pp789-796
[7] Zheng J and Feng R 2017 Real-time Video Analysis System based on Spark J. Computer System Application 26 pp53-59
[8] M. Zaharia, T. Das, H. Li, S. Shenker, and I 2012 Stoica, “Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters,” HotCloud, 12
[9] Akidau T , Schmidt E , Whittle S , et al. 2015 The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing J. Proceedings of the Vldb Endowment 8 pp1792-1803
[10] Armbrust M , Das T , Torres J and et al 2018 Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark C. the 2018 International Conference
[11] Song J, Zongzhe S, Keming M and et al. 2013 Research Progress of MapReduce Big Data Processing Platform and Algorithm J. Journal of Software 3
[12] Garcia-Gil, Diego, Ramirez-Gallego, Sergio, Garcia, Salvador and et al 2017 A comparison on scalability for batch big data processing on Apache Spark and Apache Flink J. Big Data Analytics 2 p1