Design and Implementation of Real-Time Video Big Data Platform based on Spark Streaming

Hongjun Chen*, Fuqiang Luo, Liheng Zhao and Yao Li
College of Electronic Information, Jincheng College of Sichuan University, 611731
Chengdu, China

ABSTRACT

Video data has gradually become the important component of big data from the monitoring networks, intelligent transportation networks, smart cities or other fields. In this paper, we have used the producer-consumer model of Kafka as the video streaming data acquisition layer, and the real-time processing framework Spark Streaming combined with openCV as data processing layer, and Memory, HDFS or HBase as data storage layer, and Web Technology to display the final results. During transmission process, we have used B-ASE64 encoding and JSON to implements data conversion. Therefore, the framework was designed to achieve data real-time acquisition, data processing, data transcoding, data storage and display. Meanwhile, the face recognition video capture was used as an example to build and test this framework. The relationship among the numbers of Worker nodes, batch slice and the processing efficiency was tested. The test results showed that the framework was feasible and had good real-time processing ability. And the better performance could be achieved by adjusting usage rate of CPU. This research has value for the real-time processing of image recognition, image retrieval and other big data application. But if data is deep mined, deep learning such as CNN will be needed to apply. Recently, SparkNet, a new component of Spark, has provided support for the neural network. SparkNet has made efforts to change the traditional time-consuming training for deep learning to improve greatly the efficiency.

INTRODUCTION

Multimedia such as images, audios, videos, and so on, have become the important component of big data from monitoring networks, intelligent transportation networks, smart cities or other fields. These big data often have greater value only in a short period of time when they are obtained. Therefore, the real-time acquisition and processing of video stream will be an urgent problem for big data. Spark Streaming, a big data real-time processing framework, compared with Storm and S4, has stronger integration ability and good real-time performance [1] [2].

In the paper we studied a new framework named BDRP (Big Data Real-time Processing), it solved acquisition, processing and displaying of video streaming in real time. BDRP is the framework combined Spark, Streaming and Kafka.

DESIGN FRAMEWORK

This section describes the designing method and working method of BDRP.
About Framework

This framework uses Kafka cluster to acquire data in real-time. Kafka is a message-based distributed publish-subscribe system, which has the advantages of high throughput and perfect fault-tolerant mechanism. The type of data source in this study is the video that generated by the cameras. Spark Streaming pulls messages into the cluster, and then uses Spark engine to process streaming batches, and make face detection and extract the related information combined with openCV which is a computer vision function library. After face detection and information extraction, the data can be stored or displayed directly on the Web pages. This framework is shown in Figure 1. BDRP is divided into three layers, which are data real-time acquisition layer, data real-time processing layer, and data storage and display layer [3].

![Figure 1. Framework for real-time processing of real-time video data based on Spark Streaming.](image)

DATA REAL-TIME ACQUISITION LAYER

The video streaming data was collected by the Kafka. Kafka uses the publish-subscribe message model. In Kafka, there are multiple message producers named Producer, multiple message consumers named Consumer and some shared multiple categories names Topic. In this framework, the video clients that generate data are both Producers, which push messages to the Broker that is the server agent of the Kafka cluster. The Kafka cluster contains one or more Brokers. The video stream from the cameras was read into the images sequence by openCV and encoded accordingly. The encoded images were published by Producer in message mode. The published messages were pushed to the Kafka cluster as Topics. Each class of Topic can contain multiple messages. Logically, a message can be transparently stored to one or more Brokers. A consumer that is just Spark Streaming cluster, consumed messages from Kafka Brokers by pulling named Pull. The code segments that acquired the video streaming data as follows:
Kafka manages messages by Topics. Each topic contains multiple partitions, and each partition corresponds to a logical log, and each log contains multiple segments, each segment stores multiple messages. Each partition in memory also corresponds to an index. The index records the offset of the first message of each partition. Messages in each Topic are distributed equably to multiple partitions by the method that Broker adds the messages to the last segment corresponding to one partition in the form of messages queue; the broker will create a new segment until the segment reaches a configuration value or threshold.

As mentioned above, Kafka is made up of three parts: Producer, Broker and Consumer. Kafka is a distributed messages system. The related objects of all the parts are supported by Zookeeper. All the related objects need to be registered in Zookeeper. After the broker was registered, it is saved as a temporary node in Zookeeper, which contains the corresponding broker's IP, PORT and other information. After the producer was registered, the topic is assigned automatically to one broker, one partition by Zookeeper. After the consumer was registered, the consumer is assigned a global and unique Group ID, and Consumer ID usually in the form of hostname:uuid by Zookeeper. So Zookeeper saves the relationship between Kafka cluster, and plays the role of adaptive self-management of Kafka cluster.

DATA REAL-TIME PROCESSING LAYER

In this framework, Spark Streaming consumes streaming data from Kafka and maps the data to DStream. DStream is divided into batches and transformed into RDD batches, and the batches form RDD DAG(RDD Acyclic Graph), and then form tasks after a series of jobs submission, the tasks are executed concurrently in ThreadPool of each Worker node. The task is formed as follows: each RDD is a set of Partitions, each Partition in the RDD is a Block, and each Block is a data block corresponding to one data node. The data block becomes a task after the job submission, as shown in Figure 2.

![Figure 2. Each RDD is distributed to the Worker nodes for concurrent execution.](image)
The SocketReceiver.receive function is the beginning of the receive process, which is responsible for receiving data. The received data was stored in BlockGenerator.currentBuffer. After repeated processing timeout timer in BlockGenerator was activated, the updateCurrentBuffer function encapsulated the entire data in currentBuffer, and formed new Block, and pushed the block to the blocksForPush queue, which supports FIFO. After lockGenerator BlockPushingThread works, blocksForPush queue was transferred to the BlockManager through the pushArrayBuffer function, and stored in MemoryStore. In blocksForPush queue existed many blocks, each Block contains a time stamp and the received raw data. Then PushArrayBuffer passed the blockId to ReceiverTracker by the BlockManager, and ReceiverTracker stored the blockId in the corresponding StreamId queue. ReceiverTracker also put the block that had been received but not yet processed into receiverBlockInfo. ReceiverBlockInfo is a Hashmap. The data was then extracted by the generateJobs to generate the corresponding RDD. The data processing procedures are partly as follows:

```scala
val messages = KafkaUtils.createDirectStream
  [String, String, StringDecoder, StringDecoder](ssc, kafkaParams, topics)
messages.foreachRDD { rdd => rdd.foreachPartition
  (partition =>... partition.foreach
  {record =>...})
```

In the process of data acquisition and processing, the format of data experienced a series of changes. In the Kafka cluster, the video data from the cameras was read in as the form of matrix celled Mat. In order to facilitate the transmission, the Mat data and the IP information of the camera were treated as an information object, and encoded by Base64 into the JSON object called JSonObject. JSonObject is a common information transmission format in heterogeneous network environment. The JSonObject plays a major role in the data processing, as shown in TABLE I.

<table>
<thead>
<tr>
<th>uuid</th>
<th>jsonObject</th>
<th>bootstrap.servers</th>
<th>buf.memory</th>
<th>batch.size</th>
<th>…</th>
<th>key.serializer</th>
<th>val.serializer</th>
</tr>
</thead>
</table>

If not encoded, the binary stream of Mat data is not supported in data processing. Base64 is an online picture conversion encoding system, which doesn’t change the traditional protocol, is the extension of the traditional protocol, supports all bytes transmission in binary files, suitable for network transmission of pictures. After the Spark Streaming cluster got the data stream, Base64 needs to decode the JSON object of the picture back to the Mat matrix oppose to the previous encoding process, so as to obtain the original image for face detection.

DATA STORAGE AND DISPLAY LAYER

In Spark Streaming, consumed data was cached in memory to generate two copies by default, one for data recovery, and one for computing. You could use StorageLevels.MEMORY_AND_DISK_SER’s RDD storage level by default when data entered memory. Generally, the data didn’t modify the original storage level during processing. Spark Streaming data can also be persisted to HDFS or HBase.
After the above process, the result data can also be passed directly to the Web socket monitor port. The Web used a loosely coupled structure that could be developed by any language as long as the IP and port numbers could be provided for Spark Streaming. The Web socket server accepted the result data from Spark Streaming and displayed it on some webpages at a certain frame rate.

EVALUATION
Cluster Configuration

The cluster was built on the lab OpenStack cloud platform, 6 virtual machines were created, the slave nodes Workers were created by cloning from the master node Master and modifying configuration. The cluster information was shown in TABLE II and TABLE III.

<table>
<thead>
<tr>
<th>node name</th>
<th>number of vm</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master</td>
<td>1</td>
<td>hard disk 20G, memory 8G, CPU 4 kernels</td>
</tr>
<tr>
<td>Worker</td>
<td>5</td>
<td>every one: hard disk 20G, memory 8G, CPU 4 kernels</td>
</tr>
</tbody>
</table>

Test Methods

Some parameters that affect the real-time performance of clusters were tested in this study. Data sources are real-time from cameras. Human activity videos from cameras were used as an example.

\[ T_{\text{latency}} = T_{\text{end}} - T_{\text{start}} \] (1)

Cluster real-time was measured by data delay time (referred to as latency). Tend was the last time the face is detected, Tstart was the start time of the producer's sending message. The delay time included the network transmission time, the task scheduling time and the data processing time. The shorter the delay time, the higher the real-time performance.

<table>
<thead>
<tr>
<th>soft</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux os</td>
<td>ubuntu-14.04.4</td>
</tr>
<tr>
<td>Hadoop</td>
<td>hadoop-2.6.4</td>
</tr>
<tr>
<td>Scala</td>
<td>scala-2.10.6</td>
</tr>
<tr>
<td>Spark</td>
<td>spark-1.6.1</td>
</tr>
<tr>
<td>JDK</td>
<td>jdk-8u92</td>
</tr>
<tr>
<td>Kafka</td>
<td>kafka-2.10</td>
</tr>
<tr>
<td>Zookeeper</td>
<td>zookeeper-3.3.6.tar</td>
</tr>
<tr>
<td>HBase</td>
<td>hbase-1.1.5</td>
</tr>
<tr>
<td>openCV</td>
<td>openCV-2.4.12 for Linux</td>
</tr>
</tbody>
</table>

In this test, we had 2 camera-clients. Data was distributed to Worker nodes concurrently. Due to the Yarn cluster mode, the number of Workers in the Spark Streaming cluster is at least 2. The test results were shown in Figure 3. X-axis represented the quantity of the number of Workers, and Y-axis represented the latency. This test results showed that when the number of Workers increased, the processing...
power of the cluster is stronger and the latency is shorter. But not yet the more workers, the better performance. Because if launched excessive workers, more start-up time would need and idler resources would be wasted.

![Figure 3. Number of workers impact on the latency.](image)

![Figure 4. Send-time batch slice and receive-time batch slice impact on the latency.](image)

In another test, we tested batch slice impact on the latency. Each Worker should receive an appropriate amount of processing data at each time slice. In Figure 4, we set each camera to 1280X720p HD image mode, 30fps smooth frame rate, OpenCV capture window pixel size set to 200x200p, then the amount of data in one second was about 1.01548M. We had 4 cameras, within 7s send-time batch slice, the amount of data could reach about 28.4336M. If real-time processing was not used, only one hour of continuous storage of data would reach 14.2802G.

We tested the impact of the time slice setting on the latency through 7 sets of data, shown in Figure 4. X-axis represented the group order of test data, and Y-axis represented the time. Each group had three column graphs; the first represented send-time batch slice of the messages, the second represented the receive-time batch slice of the messages, the third represented the latency. It could be seen that the sixth group had the best performance. So the batch slices were decided by the data amount.

In this test, data just was constantly "digested" in 7s. It showed that the configuration of clusters needs to tune to better performance under different amounts of data. So to get the best performance, the value of ρ should be bigger.

\[ \rho = \sum_{n=1}^{N} \sum_{k=1}^{K} C_{n,k} / (N \cdot K) \]  

(2)
N is the number of CPU cores, K is the PCU service time, t is at the t moments, n is the n CPU cores, $C^n_t$ represents the n CPU core occupied state at t moment. If $C^n_t = 0$ represents that is not occupied, $C^n_t = 1$ represents that is occupied to process data.

To improve the collection and processing performance of the platform, it is also necessary to clean up the data that is no longer in use. Since Spark Streaming uses the Spark computing engine, it is the memory-computing model, and the received data is stored in memory. Therefore, it is necessary to clean up the timeout and useless data in time to free the available memory. You can periodically clean up memory by setting reasonable spark.cleaner.ttl.

In a word, the efficiency of the frame was related to the number of worker nodes, the batch slice of sending data and the batch slice of receiving the data. The setup of the batch slice was related to the amount of the data.

CONCLUSIONS

This paper proposed video data processing framework based on Spark Streaming combined with cameras, Kafka, Spark, Streaming, openCV, etc. Kafka was used to acquire the camera data in real-time and Spark Streaming was used to process the real-time data. The cluster test results showed that the performance of the framework was affected by the number of Worker nodes and the amount of camera data and the setting of batch time. The performance can be optimized by adjusting the occupancy rate of CPU. This research has the certain significance for real-time processing applications such as image recognition and image retrieval. But this method can only extract some simple information in the video, if deep learning such as object classification, moving object detection, behavior recognition is needed, a model supporting CNN (Convolutional Neural Networks), even 3D CNN will be necessary. At present Spark has released deep learning framework SparkNet [4], which supports CNN, and makes efforts to resolve the traditional way of deep learning training extremely time-consuming problems, and improves greatly the efficiency of neural networks training.

ACKNOWLEDGEMENT

Supported by the key cultivation project of Department of Education of Sichuan Provincial (project no: 16CZ0039).

REFERENCES