AlgoLook

Uzmar Gómez

January 29, 2020

Architectures

The first reference we could find on the subject is in [1], a study on face recognition of pedestrians by analyzing live video streaming. They describe this process as computationally challenging, given that this video process involve different algorithms and steps, such as face detection, shadow removing, background substraction, feature extraction, and so on. They give the example of the face recognition algorithm, which can be executed very quickly and therefore is recommended to put it on more than one nodes in order to get the faster response. On the contrary, some heavy weight algorithms such as Gaussian mixture model for background substraction, need to be put into effect online if a system needs to process streaming video data. Therefore, in order to process video effectively and efficiently, the use of both batch processing capabilities and also real time in-memory processing capabilities is recommended. Apache Kafka deals the problem of live data transfer with high efficiency.

As for Spark, its a fast, generalised cluster-computing system. It offers an abstraction called "resilient distributed datasets" (RDDs) to support multi-pass applications that require low-latency data sharing across multiple parallel operations. RDDs can be stored in memory between queries without requiring replication. Instead, they rebuild lost data on failure using lineage: each RDD remembers how it was built from other datasets to rebuild itself.

They solve the problem of pedestrian face detection first by receiving the video stream data from a cluster of IP cameras with the video stream collector, which uses the OpenCV video-processing library to convert a video stream into frames. It then sends the messages to the Kafka broker using the KafkaProducer client. To process a huge amount of video stream data without loss, it is necessary to store the stream data in temporary storage, so this client works as a buffer queue for the data that the video stream collector produces. Keeping the data in storage before processing ensures its durability and improves the overall performance of the system as processors can process data at different times and at different speeds depending on the load.

The video stream processor is built on Apache Spark, which uses a Structured Streaming API to consume and process messages from Kafka, and uses the OpenCV library to process video stream data. A similar architecture is described on the following link.

	Training Time (s)		Recognition Time (s)	
Methods	C++	Python	C++	Python
Eigenfaces	2.10	6.70	2.83	8.45
Fisherfaces	1.07	4.75	1.89	7.32
Local Binary Pattern	1.24	4.49	1.95	7.75

Table 1: Execution times for different algorithms implemented in C++ and Python

In the article [2], the authors talk about the performance on the detection of intruders when using Apache Kafka and Spark Streaming. They mention that, when experimenting on the UNSWNB-15 dataset, this architecture has a good performance in terms of processing time and fault-tolerance on this huge amount of data.

Programming Languages

Although there is little research on the subject, there had been some studies that compare the performance when using OpenCV, either with Python or with C++.

For instance, on the article [3] the authors explore the idea of using face recognition systems to detect if a student is absent from school. They check the different times for training and recognition of Eigenfaces, Fisherfaces and Local Binary Pattern algorithms used in face recognition, using OpenCV with both Python and C++ programming languages.

They perform two experiments, one using attendance lists in educational institutions and the other with the ORL dataset. The hardware they used was an Intel's i7 2.4 GHz 4core processor, with 64-bit Windows 10 operating system. Here is the table they show as result

Another example can be viewed in the article [4]. The purpose of the authors is to implement a video alignment algorithm, which takes two videos recorded at different times on similar trajectories and possibly different speeds, and finds the best matching pairs of frames for those two videos. This algorithm may be used to detect changes between two different recordings. It was tested using a dataset of videos recorded while driving. They describe how they use Python to script their algorithm and then they rewrite it in C++, given that they want to achieve real time performance.

The machine they used was a Ubuntu Linux 14.04 with Intel Core 2 Quad CPU Q9400 at 2.66GHz, 3MB L2 cache, 8GB RAM, with GCC 4.9.2. The reference videos, captured under controlled conditions, have a duration of 76 sec, and the query videos, recorded in order to compare with the reference, have a duration of 25.1 sec.

The table 2 present the results they obtained where they found different performance between Python and C++. It's not necessary to describe each step, but is worth mentioning that they find Python to be usually

Phase	C++	Python
MSH ref. video	450.33	427.25
MSH query video	143.70	141.19
quads kd-tree build	4.55	156.58
vote	11.05	71.72
critical path matching	0.27	10.06
ECC	108.18	123.38

Table 2: Results obtain for the execution times (*s*) of different stages on a video alignment study.

slower, except on the MSH stage, where they explained that the NumPy package they used employs a highly efficient Fortran BLAS (Basic Linear Algebra Subprograms) underneath, while standard OpenCV has only a decent implementation of linear algebra routines. For all the other steps of the process, C++ has an obvious advantage on execution times.

References

[1] Abhinav Pandey and Harendra Singh. Face recognition of pedestrians from live video stream using apache spark streaming and Kafka. *International Journal of Innovative Technology and Exploring Engineering*, 7(5):4–10, 2018.

[2] May Thet Tun, Dim En Nyaung, and Myat Pwint Phyu. Performance Evaluation of Intrusion Detection Streaming Transactions Using Apache Kafka and Spark Streaming. *2019 International Conference on Advanced Information Technologies, ICAIT 2019*, pages 25–30, 2019.

[3] İlkbahar Fatih and Kara Resul. Performance Analysis of Face Recognition Algorithms. 2017.

[4] Alexandru E. Susu, Valeriu Codreanu, Georgios Evangelidis, and Lucian Petrica. Efficient implementation of a video change detection algorithm. *IEEE International Conference on Communications*, 2016-Augus:77–82, 2016.