2015 IEEE International Conference on Multimedia Big Data

An Adaptive Solution for Large-Scale, Cross-Video, and Real-Time Visual Analytics

Xiao Hu, Zhihong Yu, Huan Zhou, Hongbo Lv, Zhipeng Jiang, Xiang Zhou
Intel Asia-Pacific Research & Development, Ltd.
Shanghai, PRC
{albert.hu, zhihong.yu, huanx.zhou, hongbo.lv, zhipeng.jiang, xiang.zhou}@intel.com

Abstract— **This paper aims at a new challenge caused by a specific type of real-life problems that require to not only process tremendous videos, dig out cross-video information, but also guarantee a real-time responsiveness to the user. Moreover, users want to adapt the resources to the actual amount of visual objects, rather than to the number of videos, in order to better match resource consumption to the true business needs. All the state-of-the-art solutions cannot meet these requirements altogether, while this paper developed a series of techniques to address each specific problem, and then proposed a new adaptive solution for large-scale, cross-video, and real-time visual analytics.**

Keywords-visual analytics, real-time, cross-video, adaptability

I. Introduction

The fast trend of urbanization on the globe, especially in the emerging markets like China, is increasingly demanding sophisticated surveillance solutions, which requires to apply visual analytic techniques to real-life scenarios. Although many techniques were invented to accurately detect and recognize visual objects (text, face, etc.) in an image or video [1][2], here comes a different challenge. For example, in order to more efficiently manage the vehicles, government are eager to immediately figure out the correlations among all the vehicle occurrences across the thousands of locations in a city. This requires not only to process *massive videos*, dig out *cross-video* information, but also guarantee a *real-time responsiveness*. As explained below, the state-of-the-art solutions can't address these requirements altogether.

A. Two-Stage Solution

To process the large number of videos, many people have started to integrate Big Data technologies, e.g. Hadoop, etc., with visual analytics [3][4]. Such a solution is usually comprised of two stages. The first stage contains one or multiple front-end systems, and the second stage is a back-end centralized system. Each front-end system is responsible for transforming raw videos into tangible data (called metadata) and then sending the metadata to the back-end. Then, the back-end system will use Big Data technologies to store and analyze all these metadata. This structure is good at cross-video analytics, since all the metadata are centralized in one place. However, because the back-end system takes much longer latency (minutes, hours, or days) than the interval between the adjacent video frames (milliseconds), this type of solution cannot offer real-time responsiveness.

B. Simply Parallel Pipeline Solution

There is another type of solutions that targets real-time responsiveness. Their structures can be depicted as a bunch of pipelines. One pipeline is to handle one video, and is independent from other pipelines. Each pipeline consists of a series of stages, sequentially doing visual analytic tasks stage by stage (decoding, segmentation, feature extraction, classification, recognition, etc.). The latency of each stage is constrained no longer than the interval between the adjacent video frames, so as to guarantee real-time responsiveness. However, in this case, the customer requirements must remain relatively simple (e.g. to detect a moving object is crossing a line), because complicated requirements usually lead to time-consuming tasks (e.g. use HOG [1] algorithm for feature extraction), which would then cause the corresponding stage's latency to exceed the real-time limit. Besides, since the pipelines are isolated from each other, no information can be dig out cross the videos, which prohibits any cross-video analytics.

Besides, all the state-of-the-art solutions share another common disadvantage: the amount of system resources (e.g. number of CPU cores) have to be determined by the number of videos, no matter how many visual objects actually appear in the videos. For example, although there're much fewer vehicles during a non-traffic period versus a traffic period, the solution has to consume almost the same amount of system resources all the time. This caused end-user's concerns on cost- and energy-efficiency. They are looking for solutions *adapt to the amount of visual objects*, so that resource consumptions can match true business needs.

Also, adapting to the amount of visual objects brings another benefit: one solution can work for multiple scenarios with different numbers of videos. For instance, there is a small space with dense objects and another big space with sparse objects, while their total amount of objects are almost the same. Usually there must be more cameras to monitor the big space, and, therefore, the state-of-the-art solutions have to use extra resources for the big space. Differently, if a solution can adapt to the amount of visual objects, it can work for both scenarios without changing any resource.

This paper proposes a new solution that cannot only process a large number of videos and dig out cross-video information, but also guarantee a real-time responsiveness and also adapt to the amount of visual objects. Before presenting the details, Section 2 will describe a few real-life use cases for large-scale, cross-video, and real-time visual analytics. Then, Section 3 will elaborate the specific problems and how the proposed solution addresses each of them. Section 4 will quantitatively prove the real-time responsiveness and the adaptability to the amount of visual objects. Last, Section 5 will conclude this paper.

II. Large-Scale, Cross-Video, and Real-Time Visual Analytic Use Cases

Nowadays, due to technical difficulty, large-scale, cross-video, and real-time visual analytics are still rare on the markets. Therefore, it is necessary to justify the validity of our work by introducing a few real-life use cases first.

A. Use Case 1: Detect Fake Vehicle License Plates

The high cost of vehicle license motivates more criminals to counterfeit license plates. A straightforward method to detect such crimes is to (1) identify two vehicle occurrences having the same license plate at two locations, (2) calculate the minimal speed (= shortest-distance / time-duration), and then (3) judge if that speed makes sense to a vehicle. Applying this method to every two vehicle occurrences at every moment across massive locations will require tremendous computations. For that sake, so far this has to be done by offline or interactive (not real-time) analytics, which suffer a significant delay (days or months), preventing government from taking immediate actions.

B. Use Case 2: Measure Vehicle Speeds

Today vehicle speed is measured by specialized camera, which is more expensive than ordinary ones. This limits the scale of deployments. So, people are thinking of using the ordinary cameras for speed measurement, i.e. identify two vehicle occurrences having the same license plate at two neighboring locations, and then calculate the speed (= distance / time-duration). However, like Use Case 1, a citywide application can never be done at real-time. As a result, some effective methods can't be applied, e.g. using a roadside LED to warn the driver of overspeed.

C. Use Case 3: Trace Objects

Tracing objects (e.g. vehicles) is very useful for policemen to crack criminal cases, or even prevent crimes from happening. For example, if two vehicles share the same trace for a certain period of time, it may indicate one vehicle is following another vehicle. If the vehicle being followed is a sensitive target (e.g. a cash truck), some preventive actions should be taken. Here, obviously, real-time responsiveness is the key for such cases.

D. Use Case 4: Measure and Predict Traffics

City governments all expect to predict traffic jams as early as possible. First, the traffic at each location can be measured by real-time visual analytics; then, according to the routing relationship among these locations, one can predict the future traffic of one location based on the historical traffics of other neighboring locations.

III. System Architecture & Iplementation

The proposed solution formalizes a workflow as a DAG (Directed Acyclic Graph), of which every vertex, called *stage*, represents a task along the workflow (e.g. decoding, segmentation, feature extraction, classification, recognition, data analysis, etc.) and every edge represents the data transfer between two connected stages.

A. How to Achieve Real-Time Responsiveness

Fig. 1 illustrates a simplified real-time system with a single sequence of Inputs and Outputs. Let's define Latency[*i*] as the duration between the time when the Input[*i*] is received and the time when the corresponding Output[*i*] is generated. Then, real-time responsiveness means that, for all Input[*i*], Latency[*i*] must stay below a threshold constantly. This actually requires to meet the following criterion.

- **No Stall criterion**: There must be no stall along the whole DAG workflow all the time. That's to say, at any time, if one stage for processing an Input[*i*] was finished, the system must always have available resources to execute the next stage for the Input[*i*].

For all the stages along a DAG workflow, if each stage's latency doesn't exceed the time interval between the adjacent Inputs ($t_{stage} <= t_{in}$, see Fig.1), there would be no stall. In context of visual analytics, Inputs are video frames. So, in order to meet No Stall criterion, every stage's latency must not exceed 1/*fps*, where *fps* is the video's frame-per-second rate, which is usually 5~30. However, given a typical visual analytic task (e.g. calculate HOG [1] for a frame), it's always not feasible to shorten its execution time to 1/*fps*, which is usually 30~200 milliseconds.

To address this problem, our solution leverages Stream Analytic philosophy [5], which allocates a proper number of execution engines, called *executors*, for each stage. Each executor runs a copy of the stage's task, completely in parallel with other executors. And, sufficient CPU cores are allocated to the executors in order to ensure the parallelism.

For clarity, Fig. 2 exemplifies a DAG comprised of seven stages. The first stage is a Decoder, which receives four videos, decodes each video to a sequence of frames, and then passes the frames one by one to the next stages. There are four executors being allocated for Decoder, each of which is dedicated for one video and takes 1/*fps* to produce a frame. Following Decoder, there're two stages, HOG and HAAR, which calculate HOG and HAAR, respectively, for each frame. Assuming the execution time of HOG and HAAR is twice longer (i.e. 2/*fps*) than that of Decoder, eight executors are allocated for them, respectively. In this way, every frame, generated by Decoder, can get timely processed without suffering any stall. And the same philosophy applies to all the other stages, as shown in Fig. 2.

B. How to Handle Cross-Video Analytics

First, there must be a mechanism to identify which visual objects are correlated. For vehicle use cases, if two vehicle occurrences have the same license plate, they are correlated. Once such correlations were identified, then the cross-video analysis could be conducted, e.g. measuring vehicle speeds, tracing vehicles, etc.

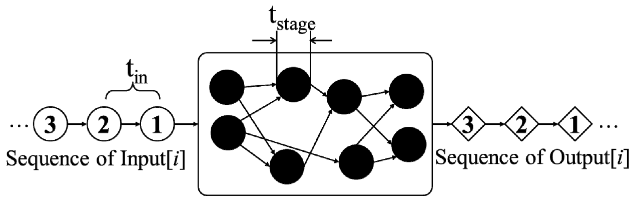


Figure 1. A Real-Time System

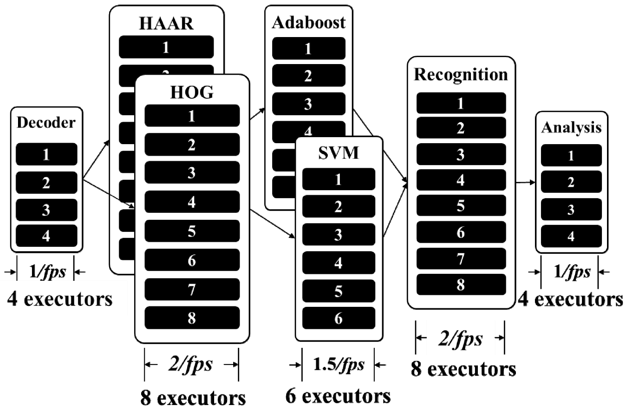


Figure 2. An Simple Illustration of The Proposed Solution

Second, since such analysis must be done at one place, there must be an inter-stage data-transfer mechanism that can move the correlated visual objects to a single executor.

For example, in Fig. 2, Recognition stage is to recognize the text string of license plates, and Analysis to conduct cross-video analysis. So, the system must know which data packets, generated by Recognition, represent the correlated vehicle objects (with the same license plate). Then, it must also know how to properly transfer the data packets from Recognition's eight executors to Analysis' four executors, so that all the correlated vehicle objects would be moved to a single executor of Analysis stage.

To achieve these, our solution utilizes another attribute of Stream Analytic technology [5], i.e. label the data packet with a *Key*. For vehicle related use cases, straightforwardly the Key is the text string of each vehicle's license plate. And, our future work will try to define more complex Keys for other use cases (e.g. the dressing pattern Key or the facial characteristics Key for pedestrian use cases).

Besides, our solution uses *Grouping Function* [5] to manage the data transfer. A grouping function specifies the mode to transfer the data packets between the executors of two connected stages. (1) It can be *shuffle* mode, i.e. a data packet can be sent to any executor of the next stage. Examples are HOG-to-SVM and SVM-to-Recognition in Fig. 2. (2) It can also be *broadcast* mode, e.g. from Decoder to both HAAR and HOG. (3) Another mode, *hash-by-Key*, is specific for cross-video analytics, where all the data packets with the same Key will be routed to the same executor of the next stage, like Recognition-to-Analysis. While this paper only handles non-ambiguous Key (e.g. license plate contains clear characters), our future work will extend to cover ambiguous Keys (e.g. license plate contains blurred characters, dressing patterns, facial characteristics, etc.).

C. *How to Adapt to the Amount of Visual Objects*

When the amount of visual objects changes, actually the workloads at different stages will change differently. Referring to Fig. 2, when there're fewer vehicles, the workload of Decoder will not change, because it majorly depends on the number of videos. However, the workload of Recognition will decrease. And, as there are even fewer correlated vehicles, the workload of Analysis will decrease even further. So, adapting to the amount of visual objects actually means to adapt such inconsistent changes of workloads among different stages. In our solution, such adaptability can be easily achieved by applying different changes to the number of executors among different stages.

D. *Putting All Things Together*

Fig. 3 illustrates the overall architecture of our solution. The bottom *Infrastructure Layer* is a cluster of distributed server nodes, each of which is equipped with CPU and GPU. Since any executor will be dynamically scheduled to any node, all the nodes should be homogeneous servers. Second, the top *Task Layer* implements the application-specific logics. E.g. detect vehicles, recognize license plate, etc.

In the middle is the *Orchestration Layer*, which is the core of our solution. It fully leverages Stream Analytics technology, e.g. Storm [5], to orchestrate a DAG workflow. In particular, the *Executor Allocation* module is responsible for allocating a proper number of executors for each logic stage. The *Grouping Function* capability is also provided by this layer, because, in this way, the upper-layer tasks don't need to care about how to transfer the data packets, but their own functionalities (Decoder, HOG, SVM, etc.). Besides, our future work will add a *Workload Adaptation* module to monitor the runtime workload of each stage, so that the number of executors could be adjusted at runtime.

The orchestration layer separates the task layer from the infrastructure layer, which makes our solution scalable. When the workload changes and infrastructure resources need be adjusted, the task layer can remain unchanged.

E. *Miscellenous*

There're a few recommendations to design a good task layer. Since the inter-stage data transfer is over TCP/IP network, it's not practicable to transfer large data packets, e.g. raw frames. Our solution fully utilizes a hardware capability, Intel Quick Sync Video [6] technology, to encode/decode the video frames with trivial overheads (<10 ms) and then transfer the encoded frames. Besides, our future work will leverage Intel CPU's compression/decompression capability to further optimize the inter-stage data transfer.

IV. EXPERIMENTAL DATA

Our experimental system implemented all four use cases described in section II. Its infrastructure layer consists of up to 8 servers. Each server has an Intel Xeon E3-1285 V3 CPU with 8 cores (4 physical cores with hyper-threading enabled) and an Intel HD P4700 processor graphics engine.

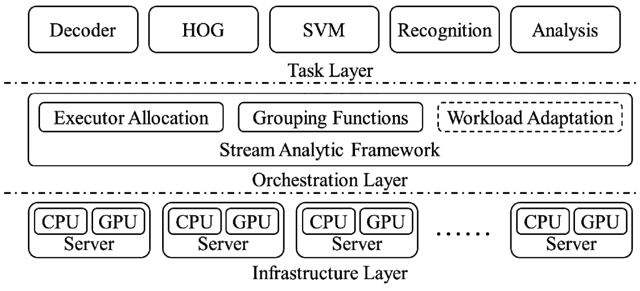


Figure 3. Solution Architecture

A. *Real-Time Responsiveness*

This experiment covered different scenarios, with the number of videos ranging from 36 to 136 and the amount of visual objects ranging from 25 to 150 vehicles / minute / video. For each scenario, Latency[*i*] were measured for each Input[*i*] (i.e. frame) to prove the system behave in a real-time manner. (Since our real-life system generates multiple Outputs for an Inputs, Latency[*i*] was defined as the time to produce the *last* Output among all Outputs.)

Fig. 4 shows the results of three scenarios, while all the other scenarios share the same conclusion. Here, X-axis lists all the values of Latency[*i*] (in milliseconds), and Y-axis shows how many times each value of latency occurs along the whole experiment. All the charts show there exist a threshold that all Latency[*i*] stay below constantly, which justifies the real-time responsiveness (see section III-A). For Fig. 4 (a), the threshold is ~2 seconds, (b) is 1, and (c) is 1.3.

B. *Adaptability to the Amount of Visual Objects*

Table 1 shows that our solution can utilize different number of CPU cores to handle different amount of visual objects, while the number of videos, 32, keeps unchanged. This proves a better cost- and energy-efficiency.

Then, Table 2 proves another advantage. Here, a bigger or smaller number of videos represents a larger or smaller space being supervised by more or less cameras. And, a smaller or bigger value of vehicles/minute/video indicates a lower or higher density of vehicles. Among these scenarios, the total amount of vehicles are almost the same, and our solution can keep using 64 cores to handle all scenarios.

V. SUMMARY

This paper proposes a solution to address large-scale, real-time and cross-video visual analytics. It eliminates the stalls along a DAG workflow by allocating the proper number of executors for each stage, to guarantee real-time responsiveness. It also introduces Keys and Grouping Functions to enable cross-video analysis. Last, it can adapt the resources to the amount of visual objects, rather than stiffly to the number of videos, which helps end-users better match their resource consumption to the true business needs.

REFERENCES

[1] Dalal, Navneet, and Bill Triggs. "Histograms of oriented gradients for human detection." Computer Vision and Pattern Recognition (CVPR) 2005. IEEE Computer Society Conference on. Vol. 1., 2005.

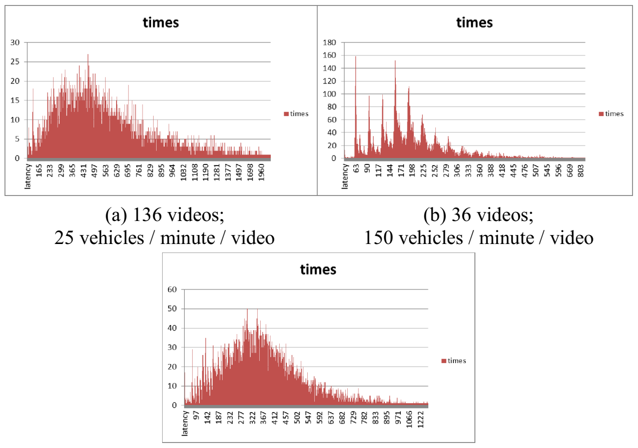
[2] Felzenszwalb, Pedro F., Ross B. Girshick, and David McAllester. "Cascade object detection with deformable part models." Computer vision and pattern recognition (CVPR), IEEE conference on., 2010.

[3] Schmidt, Rainer, and Matthias Rella. "An approach for processing large and non-uniform media objects on MapReduce-based clusters." Digital Libraries: For Cultural Heritage, Knowledge Dissemination, and Future Creation. Springer Berlin Heidelberg, 2011. 172-181.

[4] Hanlin Tan; Lidong Chen, "An approach for fast and parallel video processing on Apache Hadoop clusters," Multimedia and Expo (ICME), IEEE Conference on , vol., no., pp.1,6, 14-18 July 2014

[5] https://storm.apache.org/

[6] http://www.intel.com/content/www/us/en/architecture-and-technology/quick-sync-video/quick-sync-video-general.html



(a) 136 videos;
25 vehicles / minute / video

(b) 36 videos;
150 vehicles / minute / video

(c) 68 videos;
75 vehicles / minute / video

Figure 4. Experimental Results To Prove Real-Time Responsiveness

TABLE I. ADAPT # OF CPU CORES TO THE AMOUNT OF VEHICLES

Vehicles / Minute / Video	CPU cores needed	CPU utilization % (user)
25.0	16	86%
50.0	24	88%
75.0	32	96%
100.0	40	96%
120.0	48	96%
125.0	56	86%
150.0	64	86%

TABLE II. USE 64 CPU CORES TO HANDLE DIFFERENT SCENARIOS

Vehicles / Minute / Video	Number of Videos	CPU utilization % (user)
0.417	136	91%
0.500	128	91%
0.625	110	96%
0.830	84	94%
1.250	68	96%
1.660	50	96%
1.875	45	96%
2.000	42	91%
2.500	36	91%