Distributed Video Analysis for the Advancing Out of School Learning in Mathematics and Engineering Project

Cody W. Eilar[1], Venkatesh Jatla[1], Marios S. Pattichis[1], Carlos LópezLeiva[2], and Sylvia Celedón-Pattichis[2]
[1] {ceilar, venkatesh369, pattichi}@unm.edu
image and video Processing and Communications Lab
Dept. of Electrical and Computer Engineering
University of New Mexico, United States.
[2] {callopez, sceledon}@unm.edu
Dept. of Language, Literacy, and Sociocultural Studies
University of New Mexico, United States.

***Abstract*—The paper proposes an open-source, maintainable system for detecting human activity in video datasets using scalable hardware architectures. The system is validated by detecting writing and typing activities that were collected as part of the Advancing Out of School Learning in Mathematics and Engineering (AOLME) project. The implementation of the system using Amazon Web Services (AWS) is shown to be both horizontally and vertically scalable. The software associated with the system was designed to be robust so as to facilitate reproducibility and extensibility for future research.**

I. Introduction

There is strong interest in the development of distributed video analysis systems that can be used to analyze large video databases. Unfortunately, many methods developed for big data are not directly applicable to processing vast video databases. The paper is focused on the development of a prototype system for processing videos to understand how middle-school students from underrepresented groups learn how to program.

We begin by introducing the video analysis problem. We will concentrate on two activities as depicted in Fig. 1. Our goal is to determine whether a student is writing, typing, or not doing any of these activities. The problem can be complicated by other activities as depicted in Fig. 1(d). In any case, in this prototype system, our focus will be on developing a scalable system as opposed to finalizing the video analysis components. Currently, videos are manually analyzed [1]. The process is time-consuming, laborious, and vulnerable to human error.

The paper describes an extension of our prior research focused on the development AM-FM representations for image analysis [2], [3] and the development of dynamically reconfigurable architectures [4], [5], [6]. Instead of working on specialized methods and dedicated hardware methods, our focus in this paper is to develop scalable methods that can be applicable to thousands of hours of videos that will be generated as a part of the Advancing Out of School Learning in Mathematics and Engineering (AOLME) project. We focus on methods that reduce feature sets from Gigabytes to Kilobytes, and developing architectures that should scale to dozens or thousands of compute nodes.

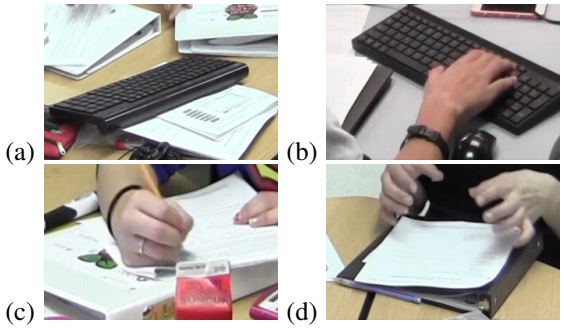

Fig. 1. Video activity detection problem. (a) No typing or writing activity. (b) Typing. (c) Writing. (d) Difficult hand communication example that is neither writing or typing.

More generally, human activity classification poses significant computational and methodological challenges. Typical features used for human activity recognition include the use of edge trajectories, optical flow algorithms, Fisher Vectors, and Gaussian mixture models [7], [8], [9]. Classification methods are commonly implemented using Linear SVM and deep learning techniques. Prior work in scaling resources on the cloud in order to minimize cost and maximize performance is discussed in [10], [11]. In this paper, we focus on the development of a system that is both horizontally (across nodes) and vertically scalable (within a node).

The rest of the paper is organized as follows. The methodology is described in section II. Results are given in section III. Concluding remarks are given in section IV.

II. Methods

A. Architecture Overview

Our system builds upon AWS to create an easy-to-maintain and easy-to-scale video processing system. We use S3 storage

to put small video clips that have been extracted from our AOLME dataset. These clips are made available to all processing nodes. The processing nodes communicate with the master node using Amazon's simple queue service (SQS). Figure 2 illustrates the basic distributed system design.

From Figure 2 we see that the first step is to upload the videos to S3. We keep the videos very small to support larger number of compute nodes and minimize latency of the overall system. On the other hand, large videos will result in a much smaller number of compute nodes that will require significant more processing and hence result in larger latencies. As we shall show in the results section, processing time scales with the size of the video. The next step is to place a message on the SQS queue specifying which video to process next. During the training phase, we also place the classification of each video segment. For our prototype system, we manually place messages on the queue so that we can control the flow. For a production system, we would have the S3 bucket notify the SQS queue that a new video was uploaded and is ready for processing. The third step is the processing step. In our setup, we create 20 Elastic Compute Cloud (EC2) instances running our feature extractor application. Each one of these instances polls the SQS queue waiting for a message to arrive. As soon as one does, it downloads the appropriate video from the S3 bucket, processes the video, and then places the results on another SQS queue. At this point, the master node is polling the results queue and collecting the results into a csv file. The csv file represents the extracted features that will be used for training the classifier. In what follows, we provide more details for each step of the system and also explain the individual video analysis components.

B. Master Node Configuration

The master node in our system is responsible for sending out jobs to process and then coalescing the results from the calculations performed by the slave nodes. The first job is to put messages on the SQS queue with a universal resource identifier (URI) that can easily be ingested by the slave nodes to download video segments. It's second job is to then poll a results queue for all the videos that were sent to be processed in the first job. The results queue contains all the features that have been extracted by the slave nodes. When the master node has asserted that all of the videos have been processed, it places the results in a comma separated value file for training the classifier.

C. Slave Node Configuration

Each slave node polls on a single queue. Once a message is received, the slave downloads the small S3 video segment, processes it using our feature extraction technique, puts the results on a queue that it has discovered on the incoming message, deletes the video locally and then begins polling on the queue again. We are able to configure our slave nodes easily using a combination of Amazon's Elastic Container Service, ECS, and Docker to distribute the software to as many nodes as we desire. We can also scale the number of nodes dynamically as more videos arrive on the queue.

D. Vertical Scalability

For vertical scalability, we want to support the use of custom architectures on FPGAs and effective implementations on GPUs. To do so, we have developed our software to take advantage of OpenCV's transparent API known as TAPI. The transparent API is an enabling technology to be able to seamlessly switch between GPU, CPU and or any hardware technologies without the software programmer having to select one at compile time or at run time explicitly. TAPI uses Open Computing Language (OpenCL) as its underlying technology to achieve significant improvements over its base algorithm suite. This fundamental technology allows the programmer to write software for a variety of hardware implementations without being burdened with implementing the algorithms by hand. Since Amazon offers a variety of node types with their service, beyond the standard horizontal scalability of the cloud, our approach supports the use of GPUs and field programmable gate arrays (FPGAs) at each slave node.

E. Distributed Feature Extraction and Classification

Feature extraction is done by the slave nodes using OpenCV leveraging the transparent API. The basic approach is to compute histograms of optical flow features and use SVM for classification.

The basic steps are as follows:

1) Decode video segment and load it into memory.
2) Calculate optical flow between two frames (either using Farneback or Lucas-Kanade methods).
3) Eliminate optical flow vectors that are less than 25% of the maximum magnitude.
4) Compute cumulative density functions (CDFs) using 25 bins for (i) X and Y centroids of connected motion vector blobs, (ii) the orientations of each blob, (iii) optical flow motion estimates in the X and Y directions, and magnitudes.
5) When all frames in the video have been calculated, place results into a comma separated value list for each video segment.

Here, after some initial testing, we settled for Farnenback's method for computing the optical flow vectors. As a result of feature extraction, we have a dramatic reduction of the input space from gigabytes down to only a few kilobytes.

Classification was much simpler than feature extraction. It required significantly fewer resources and was performed on the master node after collecting the features from each video. Initially, we performed feature selection using the Wilcoxon ranksum test. Then, the features that were found to be statistically significant were used with support vector machines.

III. Results & Discussion

A. The AOLME Dataset

The AOLME dataset is a large repository of over 900 hours of video recordings of students. The videos contain students

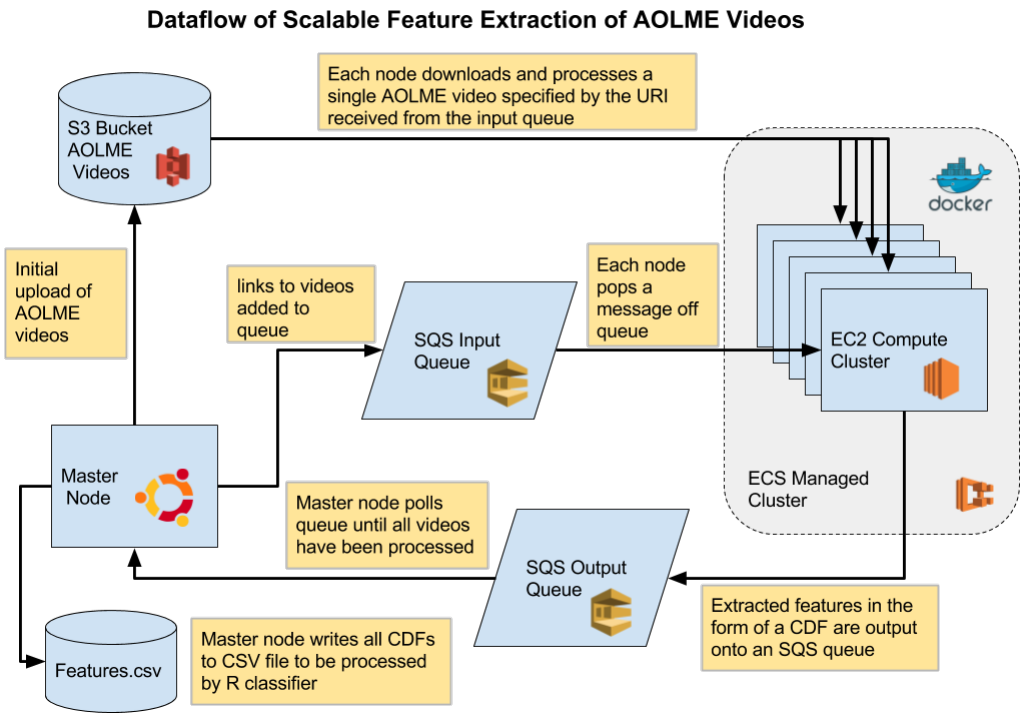

Fig. 2. Dataflow of the distributed video system using AWS components in the cloud

interacting with facilitators, their peers and computers to write code in Python on the Raspberry Pi. The videos were cropped to isolate the video activities. The basic test database included 20 video segments of typing, no typing, writing, and no writing. The basic idea was to test the detection of writing and typing activities. However, in future work, to support quantitative analysis, we will need to develop methods that: (i) detect and distinguish among multiple activities, (ii) detect the beginning and ending of each activity, and (iii) associate each activity with a particular student or group of students.

B. Classification Results

We present two sets of results. First, for typing activity detection, we performed leave-one-out cross-validation on 40 video segments (20 typing versus 20 no-typing examples). Second, for writing activity detection, we performed leave-one-out cross-validation on another 40 video segments (20 writing versus 20 non-writing examples).

For typing classification, the system performed very well. Overall, we have a classification accuracy of 90%. Based on the leave-one-out results, we have correct classification of 19 out of the 20 typing videos and 17 out of the 20 no-typing videos.

The same system gave less impressive results on writing versus no-writing activities. Overall, we have a classification accuracy of only 65%. SVM correctly classified 18 out of the 20 writing videos but mis-classified 11 no writing videos as writing. Apparently, hand gestures were incorrectly classified as writing activities.

It is clear that there is a need to further develop both the feature extraction and the classifier for more complicated human activity detection. Yet, the focus of the current paper is on developing a scalable approach as we shall describe next.

C. Proof of Scalability

We perform two experiments to demonstrate horizontal and vertical scalability. For horizontal scalability, we consider speedup as a function of the number of nodes. For vertical scalability, we consider execution time as a function of the size of the video in different hardware platforms. Furthermore, we have found that keeping the videos under 2MB gave optimal results. We also report bandwidth results.

In order to show that our system is horizontally scalable, we record the time it takes for the cluster to perform certain repetitive tasks. For the first experiment, we have the cluster operate using only a single EC2 instance, and then scale the experiment by one instance and compare how long it takes to perform feature extraction from 10 2.1MB videos. For this experiment, we used Amazon's t2.micro instance which contains 1 virtual CPU running on a high frequency Intel Xeon processor with turbo up to 3.3GHz and contains 1GB of memory. The results from this experiment are shown in Figure III-C.

For 10 videos, we have ideal speedups for 2, 5, and 10 nodes. The ideal speedups are plotted using a green line in Fig. III-C. On the other hand, we show linear speedup using a red line in Fig. III-C. To see if the expected level of performance has been reached, we divide execution times by the time it

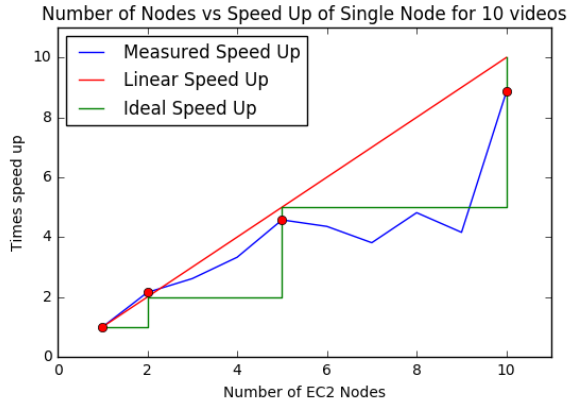

Fig. 3. Speedup as a function of the number of EC2 nodes. For this experiment, we have a total of 10 video segments of the same size (2.1MB).

takes to execute on a single node. From the results, it is clear that the measured speedups follow the ideal speedups with nearly ideal speedups for 2, 5, and 10 nodes. The speedup anomalies of Fig. III-C can be attributed to measurement variations due to possible variations in the compute nodes.

For vertical scalability, we considered different hardware platforms. We present the results in Fig. III-C. For the figure, we compare execution times for t2.micro instances against a MacBook Pro 15. The MacBook Pro had an Intel Core i7 clocked at 2.7 GHz, 16GB of RAM, and an AMD Radeon R9 M370X GPU with 2048 GB of memory. For this test, the TAPI programming model was used to automatically detect the GPU and compute the optical flow on the GPU. From the results, we cannot see execution time differences between one and ten t2.micro instances for this logarithmic scale plot. On the other hand, the GPU of the MacBook Pro performed faster but still followed the trend as function of video size.

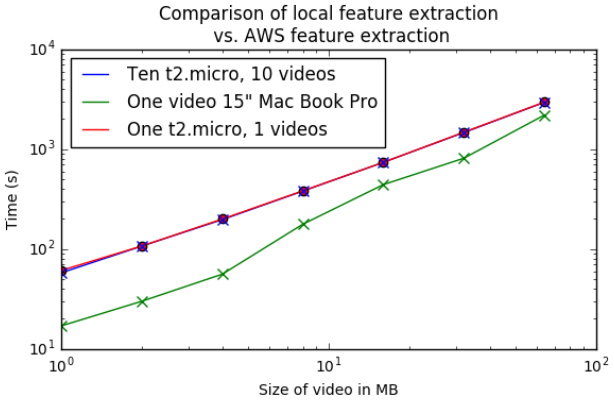

Fig. 4. Comparison of the time it takes for a single node to process 1 video vs the time it takes a cluster to process 10 videos. Videos vary in size to test the efficacy of choosing to send smaller vs larger videos to the cluster. A single t2.mirco instance was included to show that a single instance takes just as long as 10 instances.

Using several small experiments, we were able to isolate the speed of how quickly we could upload and download video segments in S3. We validated that we could achieve approximately 37 MB/s for both uploading and download videos.

D. Discussion

In the previous sections we showed that we can accurately detect typing in small video segments and demonstrated horizontal and vertical scalability. In order to achieve this, we performed feature extraction on the slave nodes and left classification to the master node. Alternatively, instead of the master node, classification could have been done on a client node. Here, we note that classification is much faster since it works with a significantly reduced feature set.

Figure III-C demonstrated horizontal scalability for at least 10 nodes. Based on the results, there is no reason to believe that this system would not scale to several hundred nodes. Nevertheless, it is possible that some minor issues may arise in achieving higher orders of scalability.

To maintain perspective, we note video classification requires a few milliseconds once the system has been trained. Also, for larger datasets that use a sufficient number of nodes so as to process no more than 10 2.1 MB video segments per slave node, the total processing time should still be completed within a few minutes.

IV. Conclusion

We presented a scalable architecture for human activity classification in video databases. The proposed method is both horizontally and vertically scalable. Horizontal scalability is achieved by cloud technologies. Vertical scalability is based on the use of OpenCV libraries that allow for easy switching between CPU and GPU implementations. The results showed that the system performed very well in detecting typing activities. Feature extraction reduced the original input video data sizes to just a few kilobytes. Because the output feature space is quite small compared with the original size of the videos input into the system, training and testing can be done very rapidly and in turn automatic classification can be done once the system has been trained at near real-time rates. The source code is available at `https://github.com/AcidLeroy/OpticalFlow`.

V. Acknowledgments

This material is based upon work supported in part by the National Science Foundation under NSF AWD CNS-1422031 and NSF AWD 1613637.

References

[1] C. LopezLevia, S. Celedon-Pattichis, and M. Pattichis, "Integrating mathematics, engineering and technology through mathematics modeling and video representations," in *13th International Congress on Mathematical Education*, Hamburg, Germany, 2016.

[2] C. Carranza, V. Murray, M. Pattichis, and S. Barriga, "Multiscale am-fm decompositions with gpu acceleration for diabetic retinopathy screening," in *IEEE Southwest Symposium on Image Analysis and Interpretation (SSIAI)*, April 2012, pp. 121–124.

[3] C. Loizou, V. Murray, M. Pattichis, M. Pantziaris, and C. Pattichis, "Multiscale amplitude-modulation frequency-modulation (am-fm) analysis of ultrasound images of the intima and media layers of the carotid artery," *IEEE Transactions on Information Technology in Biomedicine*, vol. 15, no. 2, pp. 178–188, March 2011.

[4] C. Carranza, D. Llamocca, and M. Pattichis, "Fast and scalable computation of the forward and inverse discrete periodic radon transform," *IEEE Transactions on Image Processing*, vol. 25, no. 1, pp. 119–133, Jan 2016.

[5] D. Llamocca and M. Pattichis, "Dynamic energy, performance, and accuracy optimization and management using automatically generated constraints for separable 2d fir filtering for digital video processing," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 7, no. 4, pp. 31:1–31:30, dec 2014.

[6] Y. Jiang and M. Pattichis, "A dynamically reconfigurable architecture system for time-varying image constraints (drastic) for motion jpeg," *Journal of Real-Time Image Processing*, pp. 1–17, 2014.

[7] X. Wang and C. Qi, "Action recognition using edge trajectories and motion acceleration descriptor," *Machine Vision and Applications*, pp. 1–15, 2016.

[8] H. Kuehne, J. Gall, and T. Serre, "An end-to-end generative framework for video segmentation and recognition," in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2016, pp. 1–8.

[9] J. Cai, J. Yu, F. Imai, and Q. Tian, "Towards temporal adaptive representation for video action recognition," in *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2016, pp. 4155–4159.

[10] A. S. Kaseb, A. Mohan, and Y.-H. Lu, "Cloud resource management for image and video analysis of big data from network cameras," in *2015 International Conference on Cloud Computing and Big Data (CCBD)*. IEEE, 2015, pp. 287–294.

[11] Y. Wang, W.-T. Chen, H. Wu, A. Kokaram, and J. Schaeffer, "A cloud-based large-scale distributed video analysis system," in *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2016, pp. 1499–1503.