Scalable Big Data Pipeline for Video Stream Analytics Over Commodity Hardware

Umer Ayub[1], Syed M. Ahsan[1*], and Shavez M. Qureshi[1*]
[1] Department of Computer Science, Qarshi University
Lahore, Pakistan
[e-mail : umer.ayub@qu.edu.pk; syed.ahsan@qu.edu.pk; shavez.mushtaq@qu.edu.pk]
[*]Corresponding authors: Syed M. Ahsan, Shavez M. Qureshi

Received October 8, 2021; revised December 23, 2021; accepted February 9, 2022; published April 30, 2022

Abstract

A huge amount of data in the form of videos and images is being produced owning to advancements in sensor technology. Use of low performance commodity hardware coupled with resource heavy image processing and analyzing approaches to infer and extract actionable insights from this data poses a bottleneck for timely decision making. Current approach of GPU assisted and cloud-based architecture video analysis techniques give significant performance gain, but its usage is constrained by financial considerations and extremely complex architecture level details. In this paper we propose a data pipeline system that uses open-source tools such as Apache Spark, Kafka and OpenCV running over commodity hardware for video stream processing and image processing in a distributed environment. Experimental results show that our proposed approach eliminates the need of GPU based hardware and cloud computing infrastructure to achieve efficient video steam processing for face detection with increased throughput, scalability and better performance.

Keywords: Video Analytics, Big Data, Data Pipeline, Spark, Kafka, OpenCV

1. Introduction

Recent advancement of technology has increased the amount of digital data produced every single day. Most digital data is produced by social media websites, digital video cameras, mobile devices, and various sensors connected to the Inter- net. A huge amount of this data is in the form of images and videos, providing actionable insights for businesses and individuals. Moreover, the ubiquitous deployment of cheap and readily available video cameras for security and surveillance has increased the demand for digital video stream processing and analytics [1, 2]. Traditionally the video streams are analyzed manually with the help of human operators. Depending upon the requirements, the stored video streams are viewed one by one, and the desired information is extracted. The manual inspection of video data is a time-consuming process, and human error chances are always imminent [3].

The advancement in image processing and computer vision has introduced various new paradigms for automated information extraction from images and videos. Object detection is an essential application of this field that serves as the foundation of numerous other video analytics types. Scientists have developed algorithms to classify and identify objects of interest from video without any human intervention. But the video data generated from surveillance cameras and other sources are usually massive and introduce significant challenges in analyzing and performing desired image processing operations as these algorithms are already in their nature are extremely resource-consuming [4, 5]. Therefore one of the dominant bottlenecks remains the automation of video stream processing as frame by frame video analysis requires a considerable amount of computing, network, and storage resources.

Existing solutions providing video analytics overcome this problem using expensive hardware such as Graphics Processing Units [6] and expensive cloud-based architectures that harness the power of multiple computing resources as a single unit [7]. GPU assisted video analysis techniques generally parallelize image processing tasks; therefore, significant performance gain has been observed by deploying video analytic architectures on top of GPU assisted hardware. But such hardware is usually constrained to financial limitations and extremely complex architecture level details. Similarly, there have been efforts to use cloud computing power to dissipate resource extensive, image processing tasks over multiple machines. Although cloud-based image processing systems have also shown promising results, such architectures are still constrained to security, outage, and cost. Current research in the Big Data field provides us open-source tools that can be used with commodity hardware to process a large amount of data efficiently. There is a need for a system that can harness the power of these Big Data technologies for video analytics to overcome the financial and security constraints that organizations and researchers have to face while opting for GPUs and cloud-based architectures.

In this paper, we proposed and evaluated a data pipeline system for processing video streams using open-source tools, namely Apache Spark [8], Kafka [9], and OpenCV, a well-known image processing library. Our proposed video analysis pipeline consists of three basic modules. The first module is responsible for frame extraction and their encoding to strings so that these frames can be transmitted to the messaging queue. In our second module, we used Kafka, an open-source message publishing framework, as a messaging queue that works as the Apache Spark streaming data source. The third part of our proposed pipeline is the Spark streaming application that runs on a cluster of commodity computers to apply the OpenCV algorithm on video frames in a distributed fashion and then stores the output in a database.

Our experimental evaluation shows the proposed video stream analytics pipeline's scalability and performance using different cluster sizes based on commodity machines for the

face detection task. The main contributions of this paper include:

i. Propose a video analytics pipeline using open-source big data tools that run over commodity hardware for efficient video stream processing.
ii. Integration of OpenCV image processing library with Apache Spark to simplify image processing in a distributed environment.
iii. Evaluate the proposed pipeline for face detection task using different video formats.
iv. Evaluate the Kafka producer throughput for varying image resolutions.
v. Evaluation of the proposed pipeline for scalability and performance using different cluster sizes built using commodity machines.

The rest of the paper is organized as follows. The related work is presented in Section II. In section III, we describe in detail our proposed methodology. Experimental results and performance evaluation of the video analysis pipeline is presented in Section IV. And in Section V, the conclusions and future direction of our work is discussed.

2. Related Work

The amount of video and image data being generated has increased to a tremendous amount. Not only that, manual inspection of these video streams has become impractical, but also, the commodity systems with single CPU computational power cannot process it efficiently. To overcome the challenges caused by the vast amount of data, a new technological domain is developing very fast, Big Data, which deals with heterogeneous, large-scale, and stream data. Tsai et al. [10] give a comprehensive overview of tools and programming models such as MapReduce, Hadoop, Spark, Kafka, Cassandra, HBase, MongoDB, Mahout, HDFS, Lucene, Solr, Flume, and many more to solve issues posed by Big Data [10].

Hipi-Hadoop Image Processing Interface [11] is an image processing library that is developed to be used with Hadoop Map Reduced parallel processing framework. It can also be integrated with OpenCV to perform various image processing tasks in Hadoop Map to reduce the programming environment. Hipi programs takes a HIB (HipiImageBundle) file as input, which basically is a collection of multiple images that can be represented as a single image file in HDFS. Hipi provides multiple tools and classes for creating HIBs and then writing MapReduce tasks that apply image processing algorithms on HIB in parallel and store the results to HDFS. Helly et al. [12] have discussed many tools and technologies to perform distributed image processing on large scale image datasets. Authors have described how distributed image processing can be achieved using map-reduce programming architecture and showed the application of the Canny edge detection algorithm and K-means clustering algorithm on a large dataset of images in a distributed fashion.

There have been efforts to use cloud architecture for per- forming video stream analysis. For example, in [7] Ashiq et al. proposed a cloud-based architecture to perform video stream analysis. Authors showed that how Map-reduce architecture integrated with the OpenCV image processing library can be used to perform image processing on a large number of images. The authors had also shown how the proposed architecture can achieve a significant speedup in performance when GPUs were used for processing. But the proposed architecture uses cloud architecture along with GPU assisted framework that in expensive and face network bottleneck.

Researchers have used cloud computing to overcome scalability issues posed by traditional image processing techniques. For example, in [13] B. Iqbal et al. have proposed a cloud-based system for the line detection problem using canny edge detection and Hough transformation. Similarly, Yuzhong Yan and Lei Huang [14] also propose a cloud-based solution for large scale distributed image processing using the Apache Map-Reduce programming paradigm. In their work, authors have integrated the OpenCV image processing library with Hadoop architecture and provide their cloud-based solution as PaaS (Platform as a service) for image processing researchers. Authors have also shown the results of three famous image processing algorithms, namely Discrete Fourier Transform (DFT), face detection, and template matching to show their proposed architecture's scalability and performance.

Similarly, Tingxi wen et al. [15] proposed a cloud-based medical image registration architecture that uses Spark dis- tributed architecture to accelerate medical image processing algorithms. In [16] authors have proposed an AI-based system for traffic video analysis to apply novel techniques like automated object detection, 3D reconstruction, and video tracking.

However, in such works, the cost factor and limitations of cloud architectures like security, outage, and network saturation have not been considered, and the use of commodity computers without cloud-based support is not addressed.

Researchers have made efforts to parallelize image processing tasks using the processing powers of GPUs. For example, Nan Zhang and et al. [17] have proposed a similar acceleration processing technique for image data using graphics processing units. The authors have introduced an efficient GPU architecture that gives improved computational efficiency compared to CPU. Similarly, other works such as [18] propose a Near- data-processing (NDP) architecture based on GPU power systems for image processing applications.

Most of these GPU based solutions for image processing cannot be deployed on commodity hardware. Moreover, although modern GPUs consist of many processing cores and are potentially expected to yield high performance for various applications, it is not easy to achieve ideal high throughput on such architectures as the cores of graphics processing units are grouped and data transfer among these cores is limited. Our work focuses on developing a scalable video analysis pipeline using open-source Big Data technologies on commodity computers to overcome cloud architecture's financial constraints and issues.

3. Methodology: Video Stream Analytics Data Pipeline

In this section, we give a detailed overview of our proposed methodology. Our suggested video stream analytic pipeline is illustrated in **Fig. 1**. In this technique, video analysis starts by extracting video frames from the various videos. These videos may come from many sources like websites, CCTV cameras, smartphones, and other video capturing devices.

In our proposed method to analyze every single frame of video for object detection, we introduce an integration of Apache Spark, Apache Kafka, and OpenCV that consists of the following steps:

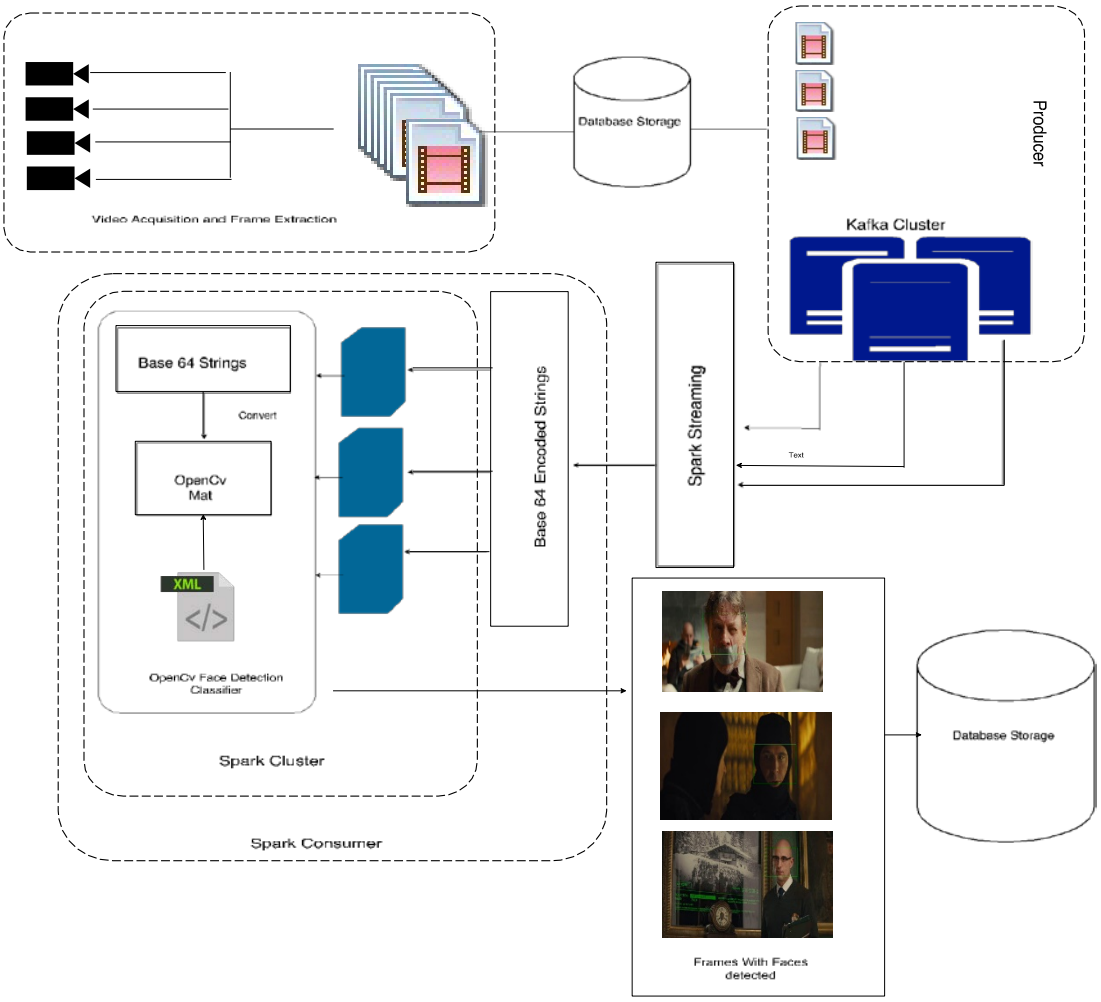


Fig. 1. Proposed solution for video stream analytics.

I. After acquiring the target video, its frames are extracted with the help of a background process written in Java language and stored in a local database.

II. Frames are then read from the disk and converted into corresponding strings to be transmitted to the Kafka messaging queue. The encoding and decoding are carried out with Xuggler [19] library that is a Java wrapper for FFmpeg [20]. We give a detailed explanation of this process in Section 3.1.

III. After encoding video frames to strings, the Kafka producer starts publishing them to the Spark streaming application. Kafka cluster is built on commodity hardware with three computer systems. Kafka distributes the incoming encoded video frames among multiple topics inside the cluster for the Spark streaming application to subscribe and process them for face detection problem.

IV. As soon as the video frames start getting published by the Kafka cluster, the Spark steaming consumer application running on a cluster of commodity computers starts to subscribe and apply the OpenCV face detection algorithm on them in parallel.

The proposed video stream analytics pipeline reads the encoded video frames from the Kafka topics, decodes strings into OpenCV mat object, detects a face on mat objects, and stores processed video frames to the local database. Complete detail of each of these steps is given in the next subsections.

3.1 Frame extraction and encoding

In our methodology, to start the analytics on a video stream, its frames need to be extracted and stored in a local database. This work has used a Java library named Xuggler for frame extraction and encoding that allows programmers to extract, encode, and decode media files directly from Java. Xuggler has been built around FFmpeg that is a C-based, comprehensive framework for media file manipulation. Compared to FFmpeg, Xuggler provides an object-oriented interface for video manipulation, and therefore, is easy to use. In our work, frames of different video formats took a varying amount of time to be extracted from video and stored in the local database. **Table 1** shows the total number of frames and resolution of each video format used as the experimental dataset.

Table 1. Experimental Datasets

	Video Resolution	**Frame rate**	**Total Frames**
Video 1 Kingsman: The Secret Service (2014)	1280 x 536	25 FPS	182,886
	704 x 480		
	352 x 240		
	176 x 120		
Video 2 Casino Royale(2006)	**Video Resolution**	24 FPS	**Total Frames**
	1920×1080		207,360
	1280 x 720		
	704 x 480		
	352 x 240		
	176 x 120		
Video 3 Harry Potter and the Deathly Hallows – Part 1 (2010)	**Video Resolution**	24 FPS	**Total Frames**
	1920×1080		210,240
	1280 x 720		
	704 x 480		
	352 x 240		
	176 x 120		

3.2 Kafka Producer

Java process responsible for publishing messages to the Kafka cluster is known as Kafka producer. Once all the video frames have been extracted, the Kafka producer client starts transmitting them to the Kafka cluster. Each video frame is read from disk and encoded to Base64 string with Xuggler API for image encoding. This process is also known as serialization. Frames are encoded to Base64 strings to be disseminated through the topics inside the Kafka brokers distributed among the commodity hardware-based cluster. For each frame encoded in a string, an instance of Kafka producer is created with suitable properties. This instance of the Kafka producer dispatches the encoded frames to the Kafka cluster as a key-value pair. Having the topic offset as key and Base64 String encoded video frame as the value. Every Kafka producer can be configured with several properties suiting the scenario. A

few of those properties are listed as following:

- **Broker list** is a comma-separated list of one or more Kafka brokers to which connection is to be established.
- **Acks property** is the number of total acknowledgments that a producer requires from the leader of the Kafka cluster before considering a request complete.
- **Max.retries** property is the total number of tries of sending a message to Kafka cluster before considering it a failure.
- **Retry.backoff** is the amount of time, the Kafka producer waits before sending a failed message to Kafka brokers again.

3.3 Kafka Brokers and Topics

Each node in the Kafka cluster is known as a broker. Each instance of the producer application publishes video frames encoded in base64 string, to a single or various virtual partitions distributed among the brokers. These partitions, grouped together, are known as the Kafka topics. We have used a Kafka cluster with three brokers and a Kafka topic for each broker with three partitions in this work. As the partitions are distributed among the brokers, the consumer application can read the records simultaneously in parallel. This distributed message publishing and subscribing characteristic of Kafka is extremely useful in developing inexpensive, scalable systems for video analytics as the power of parallelization can easily be harnessed without expensive graphics processing units. Every record in a partition has a unique identifier called off-set. As soon as our Spark consumer application comes online, it starts subscribing to the records residing in the Kafka cluster. Details of the Spark Consumer application are given in the next section.

3.4 Spark Consumer

The final module of our proposed methodology is a Spark streaming application written in the Scala programming language. This Spark consumer application is responsible for the following tasks in our proposed data pipeline:

i. Subscribing to Base64 encoded video frames residing in Kafka cluster.
ii. Decoding and converting them to a Mat object.
iii. Applying OpenCv's face detection algorithm on individual Mat object representing a frame of video, in distributed fashion at different nodes of Spark cluster.
iv. Storing the processed frames to local database.
v. Recording the face detection results.

A comprehensive detail of each part of Spark consumer application is given next.

3.5 Kafka and Spark Streaming Integration

Spark streaming application is integrated with Kafka to consume distributed records from the cluster. Kafka's simple consumer API provides the interface for this integration. Spark

streaming processes the incoming data streams by converting them to resilient distributed datasets known as RDDs in the Spark programming environment. RDDs are the basic building blocks of any Spark processing consisting of immutable objects that can be partitioned and processed among multiple computers in the Spark clusters. There are two types of configurations for the Spark streaming application to receive data from the Kafka cluster.

1. Receiver-based Approach.
2. Direct Approach (No Receivers)

In our work, we have opted for a direct approach to integrating Kafka with Spark Streaming due to its simple design and semantics. This direct approach automatically creates as many RDDs as there are Kafka topics partitions to be consumed. All these RDDs can read data from the Kafka topics in parallel, increasing parallelism's simplicity in our application. There is a one-to-one mapping between Kafka partitions and Spark RDDs.

When the Spark streaming application comes online, it starts reading video frames encoded to Base64 strings from distributed Kafka topics in parallel and creates RDDs for further processing.

3.6 Face detection in Spark Streaming Application

The records read from the Kafka partitions by the Spark streaming application and being represented by RDDs in the Spark programming environment are encoded strings that need to be converted back into an image format to apply image processing algorithms. The Spark consumer application converts these base64 encoded strings to byte arrays converted in OpenCV mat object in our proposed system. Then, the face detection algorithm is applied. We use OpenCV's LBP Cascade Classifier for face detection. There are two types of cascade classifiers shipped with OpenCV library for face detection:

1. Haar Cascade Classifier
2. LBP Cascade Classifier

We use LBP cascade Classifier for its efficiency and fast results. The Spark consumer application running on the Spark cluster processes all the records received from the Kafka topics in a distributed fashion and stores the resulting video frames in the storage of worker nodes, and records the number of faces detected in the experimental dataset.

4. Experimental Evaluation & Results

This section gives a detailed overview of the experimental evaluation of our proposed data pipeline system for video analytic on commodity hardware using open-source Big Data technologies, namely Apache Spark, Apache Kafka, and OpenCV. Our experimental setup consisted of three commodity computers running Intel Core i7 microprocessor for the Spark cluster and three Core i7 Computers for the Kafka cluster.

In the Spark cluster, we had three worker nodes, running two executors, each with a ram of 5GB assigned to a single executor. Hence a total of 30GB of ram was assigned to three worker nodes. Similarly, in the Kafka cluster, we had three machines with 8GB of RAM.

For experimental evaluation, four different resolutions, namely HD, 4CIF, CIF, and QCIF, of the Hollywood movie "Kingsman: The Secret Service," having a frame rate of 25 frames per second, are used. Each video format had a varying frame size to pixel ratio, as illustrated in **Fig. 3**, and a total of 182,886 video frames for each format were extracted, and a face detection algorithm was applied on them to check the scalability and performance gain with varying number of nodes in Spark cluster. A detailed overview of the dataset is given in **Table 2**.

Table 2. Datasets used in Experimental Evaluation

Video 1 Kingsman: The Secret Service (2014)	**Format**	**Resolution**	**Number of Frames**	**Total Size (GBs)**
	HD	1280 x 536	182,886	136
	4CIF	704 x 480		69.2
	CIF	352 x 240		21.8
	QCIF	176 x 120		6.7
Video 2 Casino Royale (2006)	**Format**	**Resolution**	**Number of Frames**	**Total Size (GBs)**
	Full HD	1920×1080	207,360	197
	HD	1280 x 536		154
	4CIF	704 x 480		78.7
	CIF	352 x 240		26.4
	QCIF	176 x 120		8.2
Video 3 Harry Potter and the Deathly Hallows – Part 1 (2010)	**Format**	**Resolution**	**Number of Frames**	**Total Size (GBs)**
	Full HD	1920×1080	210,240	200
	HD	1280 x 536		156
	4CIF	704 x 480		79.0
	CIF	352 x 240		27.8
	QCIF	176 x 120		8.9

Each individual frame's average size varied from 80 KBS to 700 KB for different video formats; also, the total numbers of pixels in a frame play an important role in image processing tasks. The large frame size indicates a greater number of pixels and higher resolution in each frame of a specific format. The varying number of pixels had a direct impact on the face detection rate and speedup in our experimental evaluation. i.e., more faces were detected in a frame with a greater number of pixels as illustrated in **Fig. 2**, but it took more time to be processed as compared to the same frame of a lower resolution. Average frame size and total number of pixels in a single frame of each format are indicated by **Fig. 3**.

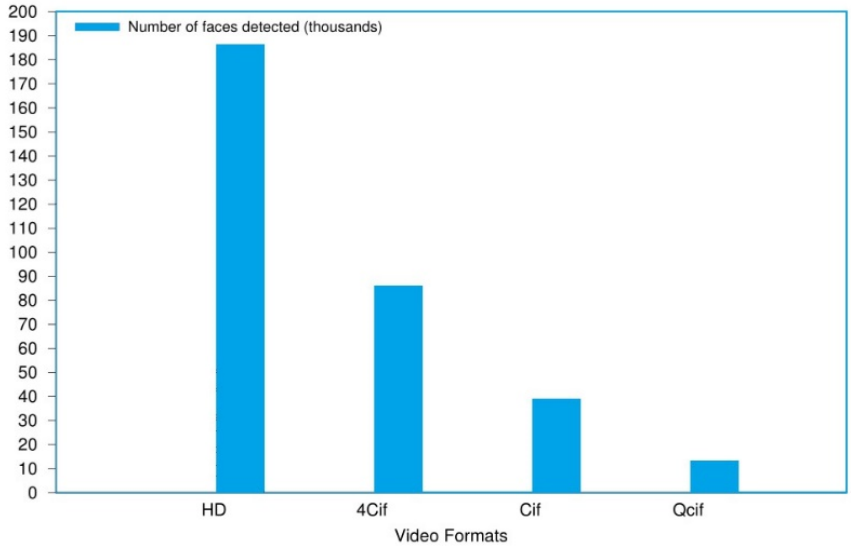


Fig. 2. Face detection: Higher resolution frames showed more face detection rate as compared to the frames of lower resolution.

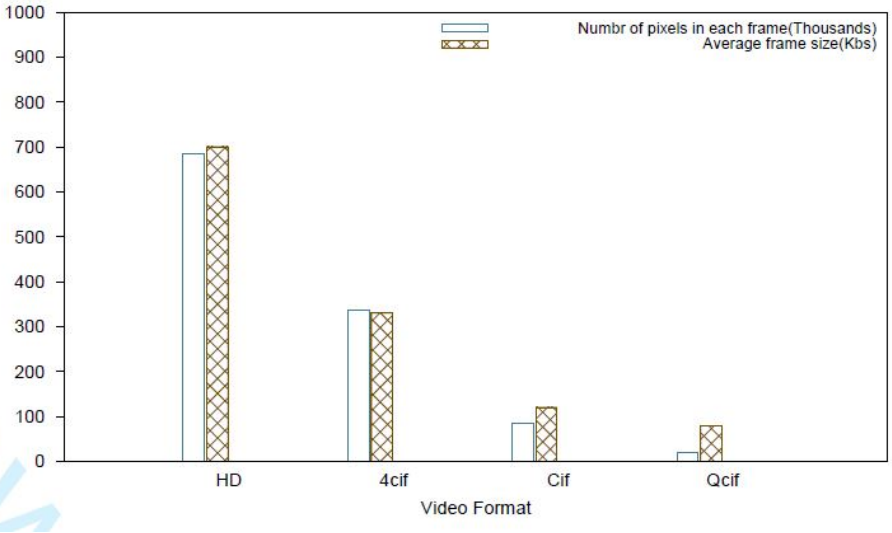


Fig. 3. Number of pixels and average frame size for different video resolutions.

4.1 Experimental Details

To evaluate our proposed data pipeline, we performed different experiments. **Table 3** describes the experiments in detail. In this experimental evaluation of our methodology for video analytics, we have examined the speed up in the thorough put of our architecture by varying the number of Spark nodes for a specific frame resolution. The total number of frames processed in one single second represents the through- put of the system. And the time taken by the system to process a single frame will determine the overall latency. In our experiments, we checked the performance gain and speed up in overall system latency and throughput for each video format by varying the number of Spark nodes in the cluster.

Table 3. Experimental Details

Experiment	Description
Experiment 01: Determining appropriate batch interval for Spark streaming.	The number of frames processed per second was compared by varying the batch interval for Spark streaming from 0.5 sec to 20 sec.
Experiment 02: Study of image processing pipeline performance.	To determine the effect of scaling on performance of pipeline, we compared the frame processing rate by changing the Spark nodes in a cluster from 1 to 3.
Experiment 03: Study of producer throughput.	Comparison of time taken by Kafka producer to encode all frames of a specific resolution and send them to Kafka topics to be consumed by consumer application.

We have used 182,886 frames of four different video formats in all experiments with a range of sizes from 6.7GB to 136GB.

In the Experiment set 01, we performed a number of iterations of experiments to determine the appropriate batch interval for Spark streaming. The detail and significance of batch interval for Spark streaming applications are explained in section titled. Experiment 01: Determining the appropriate batch interval.

In Experiment 02, we compare the time taken to process all video frames of a specific video resolution with an increasing number of Spark nodes and calculate the system's performance gain throughout. The same phenomenon can also be viewed as a performance gain in system latency as in the same experiment; we also figured the time taken by the system to process a single frame of a specific resolution. We represented it as a performance gain in the system's overall latency.

And in the Experiment set 03, we studied the performance gain in Kafka producer throughput by comparing the time taken by the producer application to encode all the frames of different resolutions and transmit them to Kafka topics to be consumed by the Spark streaming application.

4.2 Experimental Results

Here, we explain and briefly describe the results of our experimental evaluation. As explained earlier, speed up means to increase a system's performance compared to some other method, while both approaches are solving the same problem. Speed up can be measured in two ways:

1. Throughput
2. Latency

Speed up in the throughput can be measured with the help of following formula:

$$S_{\text{throughput}} = \frac{Q_2}{Q_1} \quad (1)$$

While,

- $S_{throughput}$ is throughput of System 2 with respect to System 1.
- Q_1 throughput of System 1.
- Q_2 throughput of System 2.

Similarly speed up in latency of proposed system be measured with following formula:

$$S_{latency} = \frac{L_2}{L_1} \quad (2)$$

While,

- $S_{latency}$ is latency of System 2 with respect to System 1.
- L_1 latency of System 1.
- L_2 latency of System 2.

4.3 Experiment 01: Determining the appropriate batch interval

Batch interval plays an important role in any Spark streaming application. Batch interval is the configurable amount of time, after which Spark streaming creates a batch of incoming data and starts to process it on Spark nodes. Our experiments showed that varying batch interval had a direct impact on over- all system performance. We performed a number of experiments to check the system's efficiency by changing the batch interval and selecting the interval at which the system performance was maximum. Choosing a very short batch interval caused the under-utilization of resources. Hence we increased the batch interval from 0.5 seconds to 20 seconds and checked the number of frames processed per second. **Fig. 4** indicates that the batch interval of 10 seconds showed the maximum efficiency as the system showed the maximum throughput. The results show increasing batch interval directly impacted the number of frames processed per second, but there was no significant improvement in throughput after the batch interval of 10 seconds; hence a batch interval of 10 seconds was selected.

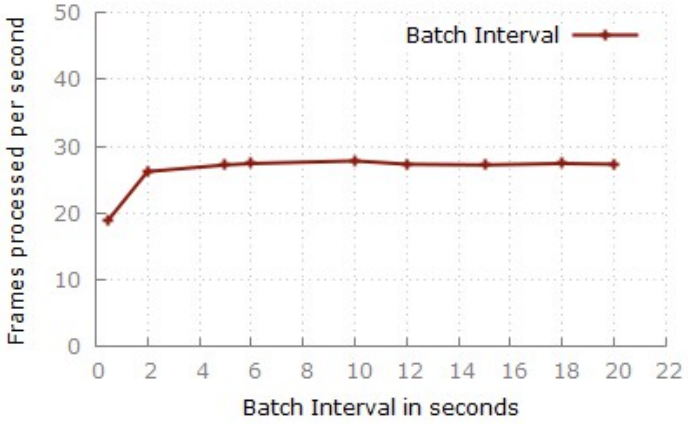


Fig. 4. Experiment 01: Determining the appropriate batch interval.

4.4 Experiment 02: Study of image processing pipeline performance

Here we explain the results of Experiment 02, where we study the speed up in throughput for each video resolution by varying the number of commodity computers in the Spark cluster. We have compared the time to process all 182, 886 frames of every video resolution and calculated the speed up in the system's overall throughput with an increasing number of Spark nodes. **Fig. 5** shows a speedup in frames processed per second for each video resolution.

The graph indicates that for video frames having the highest resolution, i.e., of 1280x536 pixels, we were able to process 28 frames per second when the Spark cluster consisted of a single worker node. When we increased the number of Spark nodes from 1 computer to 2, we were able to process 49 frames per second, and finally, having 3 nodes in Spark clusters enabled us to process 59 frames per second for HD video resolution.

Similarly, the speedup for 4CIF, second-highest resolution video format, is also shown as one Spark node-enabled us to process 54 frames per second while increasing the number of Spark nodes to 2 increased the system throughput to 93 frames per second and when a third Spark node was added to the cluster, we were able to process the frames at a rate of 102 frames per second. **Fig. 5** also indicates the speed up in throughput for CIF video format having a resolution of 352x240, as the total number of video frames processed in a second with one Spark node in the cluster is 232. In comparison, throughput increased to 340 frames per second by increasing to Spark nodes to 2, and finally, with 3 Spark nodes, we were able to process 365 frames per second, showing an increasing trend in throughput with a growing number for Spark nodes. QCIF format is the format having the lowest resolution of just 176x120 pixel per frames, and similarly having a single Spark node, we processed 357 frames per second for this video format, while increasing the Spark nodes from 1 to 2 increased the through- put to 380 frames per second. And the addition of the 3rd node increased this processing rate to 395 frames per second.

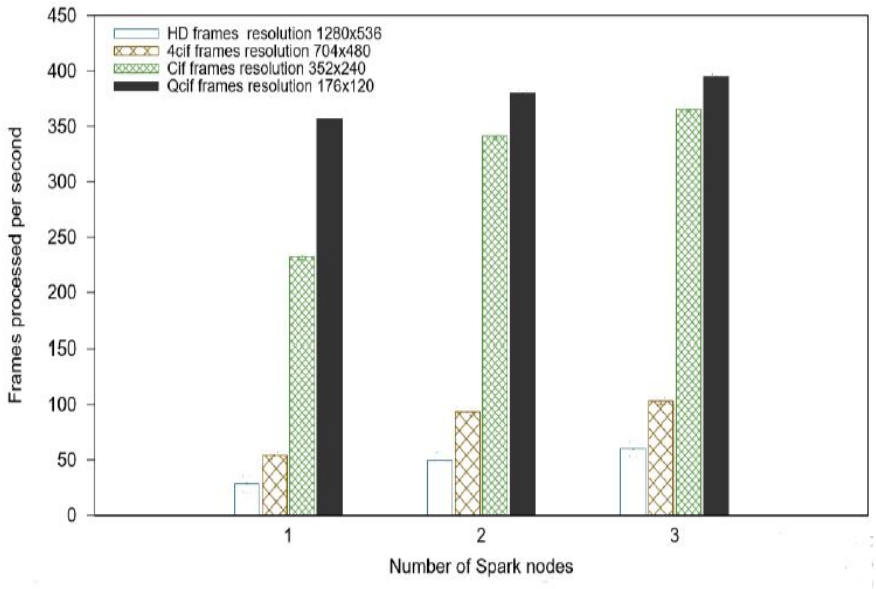


Fig. 5. Experiment 02: Speedup in throughput for different video resolution with increasing number of Spark nodes in cluster.

Fig. 6 shows a speedup in the overall latency of the proposed system. It indicates that for HD video format time taken to process a single video frame was 35 millisecond when we had only one Spark node in the cluster, but this time decreased to 20 milliseconds when we increased the number of Spark nodes to 2 and similarly we were able to process a single frame of HD format in just 16 milliseconds when Spark cluster consisted on 3 nodes. Likewise, for 4CIF video format, it took us 18.52 milliseconds to process a single frame when we had only one Spark node in the cluster, adding another node to the cluster decreased the time taken to process a single frame to 10.75 seconds and finally, the addition of a 3rd Spark node reduced the latency to 9.8 milliseconds. In the case of CIF video format, the time taken to process a single frame was calculated to be 4.31 milliseconds that decreased to 3 milliseconds when 2nd Spark node was added to the cluster, and the addition of the 3rd Spark node decreased the time to process a single frame to 2.74 milliseconds as shown in **Fig. 6**. In the case of QCIF video format, the time to process a single frame was calculated to be 2.8 milliseconds that was reduced to 2.63 milliseconds when a second Spark node was added to the cluster. Finally, the addition of a third Spark node reduces this time to 2.53 milliseconds.

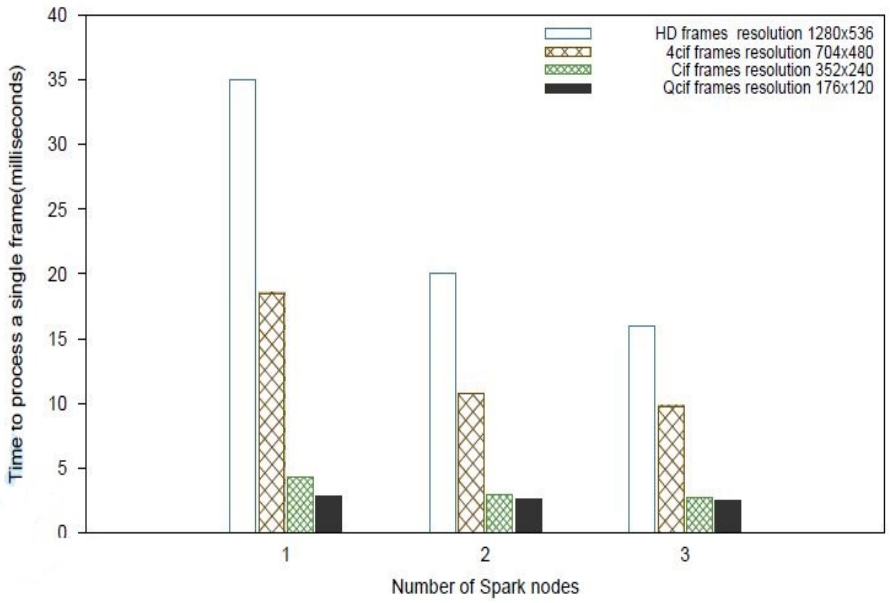


Fig. 6. Experiment 02: Speedup in system latency with increasing number of Spark nodes in cluster.

4.5 Experiment 03: Speed up in throughput for frame production for different types of frame resolutions

In this section, we explain the results of our third experiment in which we compared the number of frames encoded to string and sent to the Kafka cluster in a single second to be consumed by consumer application and calculated the speedup gained by changing the video resolution from HD to QCIF. It was observed that for higher resolutions having a large number of pixels, it took more time to encode them into strings and send them to the Kafka cluster compared to lower resolution formats. **Fig. 7** shows that for HD video format having a resolution of 1280x536, we were able to achieve a sending rate of 71 frames per second. For 4CIF video format having a resolution lesser resolution than HD, i.e., 704x480 pixels, this sending rate increased to 238 framed per second. For CIF format having a resolution of

352x240 pixels, this sending rate showed an increasing trend as we were able to encode and send video frames at a rate of 500 frames per second as shown in **Fig. 7**. Finally, for the video format having the lowest video resolution, this sending rate increased to 750 frames per second.

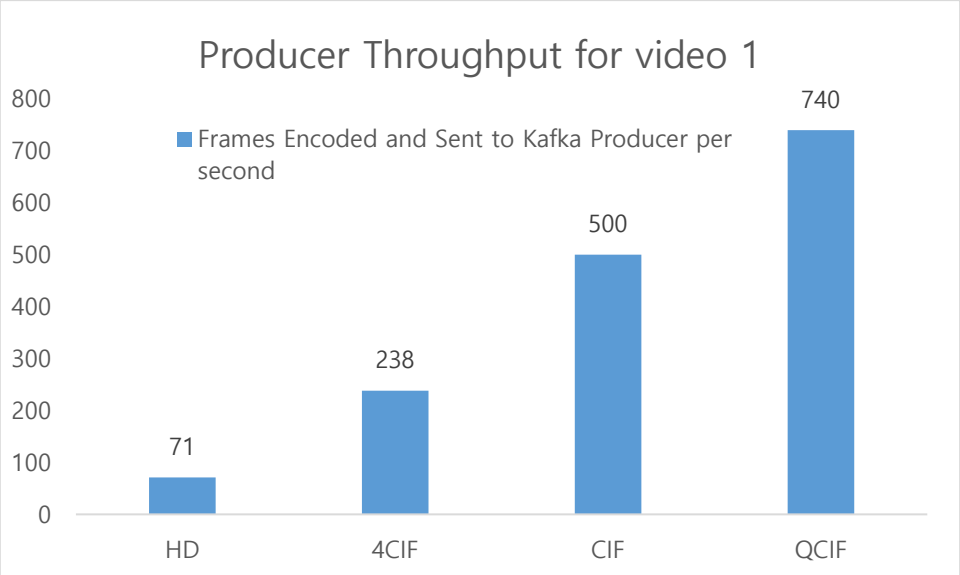
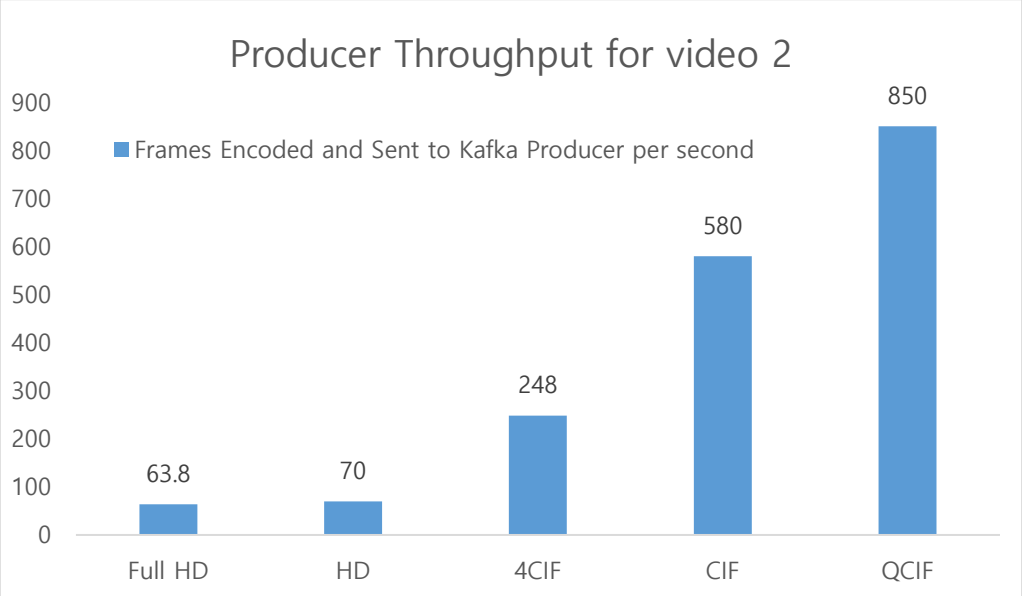


Fig. 7. Experiment 03 (Speedup in throughput) results for video 1: King's man the Secret Service (2006)



.

Fig. 8. Experiment 03 (Speedup in throughput) results for video 2 Casino Royale (2006)

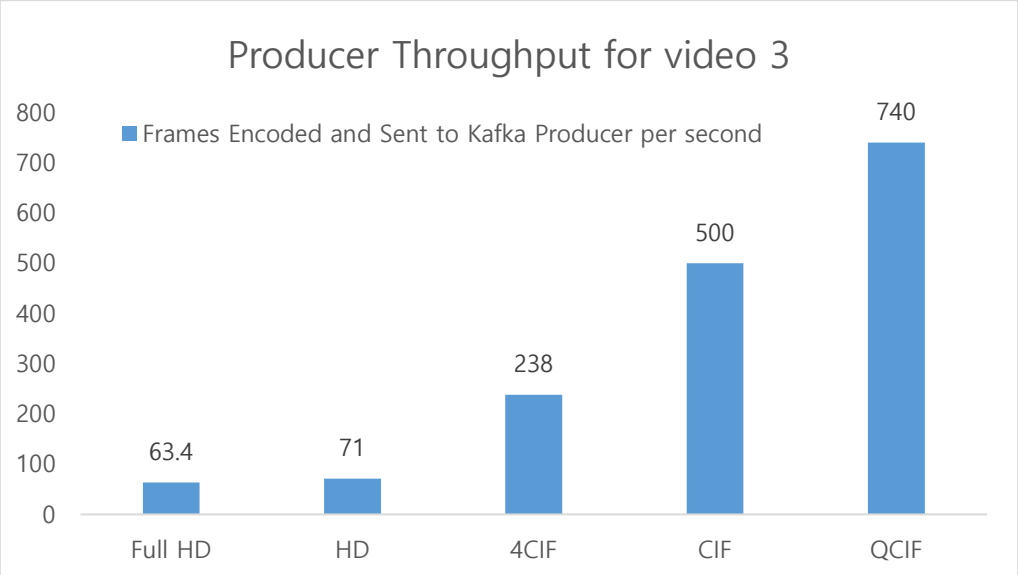


Fig. 9. Experiment 03 (Speedup in throughput) results for video 3
Harry Potter and the Deathly Hallows – Part 1 (2010)

In **Table 4**, we summarize the experimental results for different video resolution, the number of nodes used, frames processes per second, throughput, and the total number of faces detected.

Table 4. Experimental Summary

Video 1 Kingsman: The Secret Service (2014)				
Resolution	**Number of Nodes**	**Frames Processed per Second**	**Producer Throughput (fps)**	**Number of Faces Detected**
HD 1280x536	Stand Alone	3.6	-	186,886
	1 Spark Node	28	71	
	2 Spark Node	49.8		
	3 Spark Node	59.8		
4CF 704x480	Stand Alone	7	-	86662
	1 Spark Node	54	238	
	2 Spark Node	93		
	3 Spark Node	102		
CIF 352 x 220	Stand Alone	30	-	39518
	1 Spark Node	232	500	
	2 Spark Node	340.9		
	3 Spark Node	365.8		
QCIF 176 x 120	Stand Alone	128	-	12922
	1 Spark Node	357	750	
	2 Spark Node	380		
	3 Spark Node	395		

Video 2 Casino Royale (2006)				
Resolution	**Number of Nodes**	**Frames Processed per Second**	**Producer Throughput (fps)**	**Number of Faces Detected**
Full HD 1920×1080	Stand Alone	2.2	-	218,790
	1 Spark Node	22	63.8	
	2 Spark Node	36		
	3 Spark Node	47		
HD 1280x536	Stand Alone	3.6	-	186,886
	1 Spark Node	28	70	
	2 Spark Node	49.8		
	3 Spark Node	59.8		
4CF 704x480	Stand Alone	7	-	86662
	1 Spark Node	54	248	
	2 Spark Node	93		
	3 Spark Node	102		
CIF 352 x 220	Stand Alone	30	-	39518
	1 Spark Node	232	580	
	2 Spark Node	340.9		
	3 Spark Node	365.8		
QCIF 176 x 120	Stand Alone	128	-	12922
	1 Spark Node	357	850	
	2 Spark Node	380		
	3 Spark Node	395		
Video 3 Harry Potter and the Deathly Hallows – Part 1 (2010)				
Resolution	**Number of Nodes**	**Frames Processed per Second**	**Producer Throughput (fps)**	**Number of Faces Detected**
Full HD 1920×1080	Stand Alone	2	-	219,580
	1 Spark Node	22	63.4	
	2 Spark Node	36		
	3 Spark Node	47		
HD 1280x536	Stand Alone	3.6	-	186,886
	1 Spark Node	28	71	
	2 Spark Node	49.8		
	3 Spark Node	59.8		
4CIF 704x480	Stand Alone	7	-	86662
	1 Spark Node	54	238	
	2 Spark Node	93		
	3 Spark Node	102		
CIF 352 x 220	Stand Alone	30	-	39518
	1 Spark Node	232	500	
	2 Spark Node	340.9		

	3 Spark Node	365.8		
QCIF 176 x 120	Stand Alone	128	-	12922
	1 Spark Node	357	740	
	2 Spark Node	380		
	3 Spark Node	395		

5. Conclusion and Future Work

This paper proposed a system to develop a scalable, low-cost video analytic data pipeline by using open-source Big Data technologies on commodity hardware and evaluating our work by performing various experiments. Our solution integrates the OpenCV library with Apache Spark and Kafka and provides good scalability on the commodity hardware. Our approach eliminates the need for using expensive GPU-based hardware and cloud computing infrastructure for face detection video analytics. Our evaluation shows that increasing the Spark nodes in the Spark cluster increased the system's throughput. Although there is an appropriate batch interval to increase the number of frames processed per second, but this increase in throughput is limited by an upper limit on batch interval. We also concluded that for higher resolution images, the face detection algorithm's performance was better compared to images of lower resolution. In the future, we intend to develop a dynamic resource allocation algorithm for Spark and Kafka to increase or decrease the available resources depending on the demand for video streams.

References

[1] Qingyang Zhang, Hui Sun, Xiaopei Wu, and Hong Zhong, "Edge video analytics for public safety: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1675–1696, 2019. Article (CrossRef Link)

[2] Samvit Jain, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, and Joseph Gonzalez, "Scaling video analytics systems to large camera deployments," in *Proc. of the 20th International Workshop on Mobile Computing Systems and Applications*, pp. 9–14, 2019. Article (CrossRef Link)

[3] Martin Gill and Angela Spriggs, *Assessing the impact of CCTV*, vol. 292, Home Office Research, Development and Statistics Directorate London, 2005.

[4] I. K. Park, N. Singhal, M. H. Lee, S. Cho, and C. Kim, "Design and performance evaluation of image processing algorithms on GPUs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 1, pp. 91–104, 2011. Article (CrossRef Link)

[5] Michael Zink, Ramesh Sitaraman, and Klara Nahrstedt, "Scalable 360° video stream delivery: Challenges, solutions, and opportunities," *Proceedings of the IEEE*, vol. 107, no. 4, pp. 639–650, 2019. Article (CrossRef Link)

[6] J. Fung and S. Mann, "Using graphics devices in reverse: GPU-based image processing and computer vision," in *Proc. of 2008 IEEE International Conference on Multimedia and Expo*, pp. 9–12, 2008. Article (CrossRef Link)

[7] Ashiq Anjum, Tariq Abdullah, Muhammad Tariq, Yusuf Baltaci, and Nick Antonopoulos, "Video stream analysis in clouds: An object detection and classification framework for high performance video analytics," *IEEE Transactions on Cloud Computing*, vol. 7, no. 4, pp. 1152-1167, 2019. Article (CrossRef Link)

[8] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica, "Spark: Cluster computing with working sets," in *Proc. of the 2Nd USENIX Conference on Hot Top- ics in Cloud Computing (Berkeley, CA, USA), HotCloud'10*, pp. 10–10, 2010.

[9] J. Kreps, N. Narkhede, and J. Rao, Kafka: "A distributed messaging system for log processing," in *Proc. of 6th International Work- shop on Networking Meets Databases (NetDB)*, Athens, Greece, 2011.

[10] Chun-Wei Tsai, Chin-Feng Lai, Han-Chieh Chao, and Athanasios V. Vasilakos, "Big data analytics: A Survey," *Journal of Big Data*, vol. 2, no. 21, pp. 13-52, 2015. Article (CrossRef Link)

[11] Chris Sweeney, Liu Liu, Sean Arietta, and Jason Lawrence, "Hipi: a hadoop image processing interface for image-based mapreduce tasks," *Chris. University of Virginiam*, vol. 2, no. 1, pp. 1–5, 2011.

[12] Helly M Patel, Krunal Panchal, Prashant Chauhan, and MB Potdar, "Large scale image processing using distributed and parallel architecture," *International Journal of Computer Science and Information Technologies*, Gujrat, India, vol. 6, no. 6, pp. 5531–5535, 2015.

[13] Bilal Iqbal, Waheed Iqbal, Nazar Khan, Arif Mahmood, and Abdelkarim Erradi, "Canny edge detection and hough transform for high resolution video streams using hadoop and spark," *Cluster Computing*, vol. 23, no. 1, pp. 397–408, 2020. Article (CrossRef Link)

[14] Lei Huang Yuzhong Yan, "Large-scale image processing research cloud,".

[15] Tingxi Wen, Haotian Liu, Luxin Lin, Bin Wang, Jigong Hou, Chuanbo Huang, Ting Pan, and Yu Du, "Multiswarm artificial bee colony algorithm based on spark cloud computing platform for medical image registration," *Computer Methods and Programs in Biomedicine*, vol. 192, 105432, 2020. Article (CrossRef Link)

[16] M. Cao, L. Zheng, W. Jia, and X. Liu, "Joint 3D reconstruction and object tracking for traffic video analysis under iov environment," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 6, pp. 3577-3591, 2021. Article (CrossRef Link)

[17] Nan Zhang, Yun-shan Chen, and Jian-li Wang, "Image parallel processing based on GPU," in *Proc. of 2010 2nd International Conference on Advanced Computer Control*, vol. 3, pp. 367–370, 2010. Article (CrossRef Link)

[18] J. Choi, B. Kim, J. Jeon, H. Lee, E. Lim, and C. E. Rhee, "Poster: Gpu based near data processing for image processing with pattern aware data allocation and prefetching," in *Proc. of 2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 469–470, 2019. Article (CrossRef Link)

[19] A Xuggler, Xuggle, "xuggler api," 2012. [Online]. Available: http://www.xuggle.com/xuggler/

[20] Suramya Tomar, "Converting video formats with ffmpeg," *Linux Jour- nal*, vol. 2006, no. 146, p. 10, 2006.

Umer Ayub received his Master of Philosophy (MPhil) in Computer Science degree from Punjab University College of Information Technology (PUCIT), University of the Punjab, Lahore, Pakistan. He is currently serving as Full-Time Faculty Member at Department of Computer Science, Qarshi University, Lahore. His core research interests include Cloud Computing, Big Data and Network Security.

Dr. Syed Muhammad Ahsan received his PhD in Computer Science from University of Engineering & Technology (UET), Lahore, Pakistan in 2008. At present, an Associate Professor, and Chairperson of Qarshi University, Dr. Syed Ahsan has more than 26 years of academic and industrial experience. As Associate Professor, he served University of Engineering and Technology, Lahore for more than 20 years. Later he served as Associate Professor (Tenured) at King Abdulaziz University, Jeddah, Saudi Arabia for five years. His area of expertise includes Distributed Databases, Cheminformatics / Bioinformatics, Security Informatics, Cyber Security, Distant Learning, Ontologies, and Semantic Web.

Shavez Mushtaq Qureshi received his Master of Philosophy (MPhil) in Computer Science degree from Punjab University College of Information Technology (PUCIT), University of the Punjab, Lahore, Pakistan. He is working as Full-Time Faculty Member at Department of Computer Science, Qarshi University, Lahore. His research interest focuses on Wireless Sensor Networks, Cloud Computing, Cyber Security and Computer Networks.