A New Big Data Architecture for Real-Time Student Attention Detection and Analysis

Tarik Hachad[1], Abdelalim Sadiq[2]
Laboratory of Information Modelling and Communication Systems, Faculty of Sciences
Ibn Toufail University
Kenitra, Morocco

Fadoua Ghanimi[3]
Laboratory of Technological Information and Modeling
Faculty of sciences Ben M'sick
University Hassan II Casablanca
Morocco

***Abstract*—Big Data technologies and their analytical methods can help improve the quality of education. They can be used to process and analyze classroom video streams to predict student attention, this would greatly improve the learning-teaching experience. With the increasing number of students and the expansion of educational institutions, processing and analyzing video streams in real-time become a complicated issue. In this paper, we have reviewed the existing systems of student attention detection, open-source real-time data stream processing technologies, and the two major data stream processing architectures. We also proposed a new Big Data architecture for real-time student attention detection.**

***Keywords*—Attention detection; big data analysis; stream processing; real-time processing; Apache Flink; Apache Spark; Apache Storm; Lambda architecture; Kappa architecture**

I. Introduction

Student attention plays a significant role in the teaching-learning operation. It allows the student to focus on information and ignore any disturbing or distracting factor. The teacher can easily and in a natural way know if a student is in a state of attention or not. In small classrooms, students are naturally more engaged than in the largest one with a large student population or in an amphitheater. Indeed, a small classroom allows an environment that promotes student engagement as long as it is easier to monitor. In contrast, the more the classroom is large it influences the students' attention. The teacher will have to spend more time to draw students' attention and ended up losing control over part of the students. The automation of the continuous detection and evaluation of the student's attention during the lecture is the optimal solution for large audiences. In fact, it offers the teacher the possibility of knowing the attention level of the students at any time during the course. It can also notify the teacher of students with a very low level of engagement or those who are lost during the course session. Like that, the teacher can send corrective messages to less engaged students or review the course so that can be more attractive. In a previous work, we set up an architecture for the detection and the analysis of the student's attention through the use of different technologies of facial and body expressions detection. The system has been designed for monitoring a classroom with a limited number of students. It is based on the analysis of the video stream generated by a high definition camera placed in front of a classroom. The axes of analysis used are facial expressions, gaze direction, and body gestures. The analysis results of these axes are merged to deduce the level of attention of each student. The results must be obtained in real-time, for this, we opted for a parallelized computation. The analysis tasks are time-consuming, especially for a high definition image stream with a high frequency of 30 images/second. The generalization of this system on an entire school or a university will explode the number of images received by the system, that must be processed simultaneously and the output results must be provided in real-time. This high scaling requires the use of Big Data technologies in order to overcome these issues.

The main objective of this article is to present a state of the art of existing student attention detection systems and some concepts of Big Data. We also present in detail and with a comparison the different tools and architectures allowing real-time stream processing. Finally, we propose our own architecture based on the comparisons made.

II. State of the Art

A. Existing Systems

Most of the works on detecting student attention has focused on the concept of attention and its relationship to different facial and body features.

Whitehill et al. [1] based their study on the analysis of facial expressions. The goal of their work is the development of an automated system for real-time recognition of student engagement.

Krithika et al. [2] have worked on the analysis of student concentration in an online learning environment. The main idea of their work is to be able to predict the level of concentration of the student from two measurements namely the rotation of the face and the eyes' movements.

Zaletelj et al. [3] admit that the problem of detecting student attention was to establish the correct correlation between the student's attention and the teacher's observations. For this, they have designed a system that uses the capabilities of the Kinect One sensor. This device makes it possible to collect behavioral data from students in a non-intrusive way. They proposed a method to match the features of the data collected to the students' facial and body expressions. Then, they applied machine learning methods to build models for predicting student's attention.

Vettivel et al. [4] tried to establish the relationship between attention and human parameters such as heart rate variation, facial expressions, and brain waves. They used appropriate sensors to collect information on the three student parameters during the course. The student is alerted whenever he loses his concentration. This solution combines the three parameters to increase the accuracy of the system. The features are extracted from the collected data then they are classified to predict the attentive and non-attentive states.

Goldberg et al. [5] developed a manual rating instrument, to continuously measure the observable behavior of students. They used then computer vision techniques to perform automated analysis of video recordings to extract features of the students' head pose, gaze direction, and facial expressions. Using these extracted features, they tried to estimate manually annotated attention levels for each student. As they opted for continuous labeling, a regressor is trained to relate the visible features to the manual labels. For more precision, they took into account the synchronous behavior of the neighboring students.

B. Big Data

In education, Big Data technologies can be used to collect and analyze huge amounts of information about students in order to develop more effective learning. This makes the experience more practical, especially for large establishments with classes whose size is constantly increasing.

Big Data is an industrial term that was coined to describe huge volumes of data that we have never had or processed before. It also brings together techniques for storing, analyzing, and visualizing the results obtained from more varied and complex massive data structures.

The International Data Corporation (IDC) defines the Big Data as a new generation of technologies and architectures designed to economically extract value from very large volumes of a wide variety of data by enabling high velocity capture, discovery, and/or analysis [6].

According to Jason Bloomberg [7] Big Data is a massive volume of both structured and unstructured data that is so large that it's difficult to process using traditional database and software techniques.

In accordance with the Gartner definition of Big Data, which articulates its definition in three parts: Big Data is high-volume, high-velocity and high-variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making [8][7].

Big Data is a very broad concept that includes three essential dimensions: volume, variety, and velocity. This requires a real revolution in the methods of storage and data processing. Fig. 1 highlights the constraints raised by the Big data.

The volume which designates the size of the data is now greater than terabytes and petabytes. The rapid transition to these scales greatly exceeds the traditional storage and processing capacities[9][10][11].

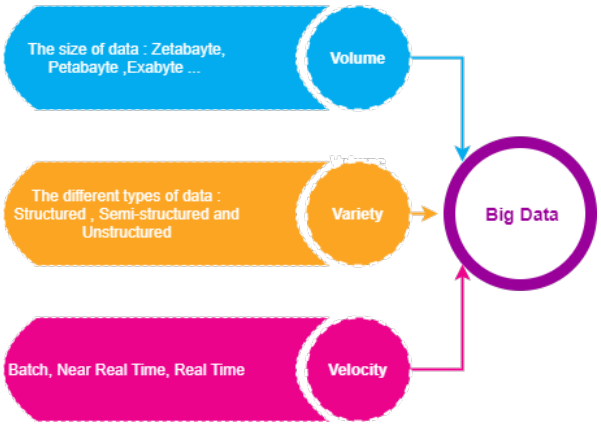

Fig. 1. The Three vs of Big Data.

Variety: This characteristic amplifies the challenge of Big Data. Since we must not only manage structured data but also semi-structured and mainly unstructured data. The vast majority of Big Data is in unstructured or semi-structured formats, such as text files, log files, audio, image, video files, social media updates, machine data, and sensors signals, etc. [9][11].

Velocity refers mainly to the speed at which data is being generated, produced, created, or refreshed. It is required not only for Big Data, but also for data processing. For time-limited processes, Big Data should be used as it streams into the organization in order to maximize its value [9][10][11].

Big Data does not primarily mean a huge amount of data or a database, they constitute the core set of technologies and components for large-scale data processing and analysis. Regardless of the type of data (structured, unstructured, or semi-structured), the data can be in one of the following three states: Data at Rest, Data in Motion, and Data in Use. Each of the formats previously mentioned requires specific processing methods.

In this paper, we deal with a case of processing data in motion, since the data is a stream of images from multiple high definition cameras. In what follows, we will present certain data stream processing tools with a comparison in order to choose the tool that best satisfies our need and that performs real-time processing.

III. Stream Processing Platforms

There are many architectures and platforms to choose from. However, selecting the right architecture with the right implementation is often difficult. In what follows we will present the most efficient stream processing tools with the details of their features as well as the most used Big Data architectures.

A. Apache Flink

Apache Flink [7] is an open-source platform that came from Berlin TU University. It supports both batch and stream processing and can guarantee an exactly-once-processing. The Flink cluster architecture is illustrated in Fig. 2. Flink is scalable, has an in-memory option, and provides input APIs in

Scala and Java. It can be integrated into the Hadoop ecosystem (HDFS, YARN), or run in a completely independent way since it has its own runtime. The core of Flink is a distributed streaming dataflow, accepting programs structured as graphs (JobGraph) of activities that produce and consume data. Each JobGraph can be executed using one of the different distribution options available for Flink (like single JVM, YARN, or cloud) [8]. Flink's processing model applies transformations to parallel data collections [9][10][11].

DataStreams represent the abstraction of Flink streams. They are similar to Storm tuples, in the form of partially ordered recording sequences. DataStreams are fed by data from different external sources such as log files, message queues, etc. DataStreams support multiple operators such as map, filtering, and reduction in a parallelized way.

FlinkML has its own machine learning library. Moreover, it provides an adapter for the SCALABLE ADVANCED MASSIVE ONLINE ANALYSIS (SAMOA) library, which offers a variety of machine learning algorithms for stream processing.

Flink fault-tolerance approach is based on snapshots over distributed checkpoints that maintain the status of jobs. Snapshots act as consistency checkpoints to which the system can return in case of failure [12].

B. Storm

Apache Storm [13] is an open-source platform for real-time stream processing, written in Java and Clojure. The origin of this project goes to the Back type company specialized in the analysis of social media data and which was sold later to Twitter. The project became an Apache top-level project in September 2014 [14]. The main data structure in Storm is the tuple, a list of serializable values that are user-defined types. Fig. 3 explains the architecture of Storm.

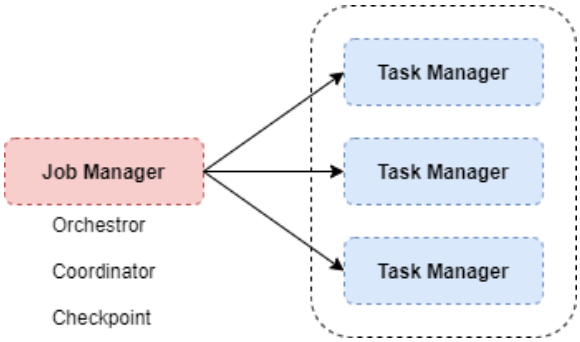

Fig. 2. Architecture of Flink Cluster.

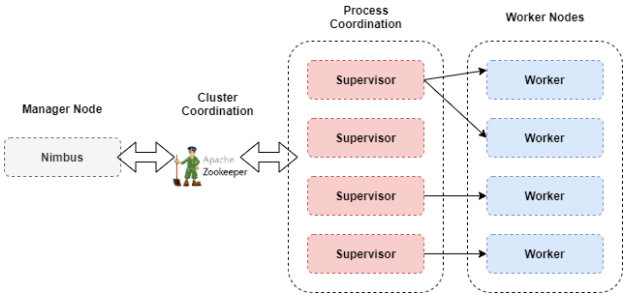

Fig. 3. Storm Cluster Architecture.

The backbone of Storm Architecture is spouts and bolts. A spout is the entry point to the stream, it connects to an external data source such as a message queue and retrieves the data in tuple format. Stream processing is performed by bolt nodes. Each bolt performs a transformation of limited complexity, in this way, several bolt nodes are grouped in a coordinated manner to perform the entire computation. A Storm application can be defined through a topology of spout and bolt nodes forming a directed acyclic graph (DAG), with arcs representing streams of tuples flowing from one node to another [15].

Apache Storm handles fault tolerance through its backup and acknowledgment (ACK) mechanism imposed by its topology. To ensure that the tuples are re-processed after failure, the spouts will always keep the tuples in their output queues until the bolts acknowledge them. When a tuple enters the topology, the spout which receives it adds an identifier, then sends this identifier to the acker bolt. The aker bolt is as a register for all tuples Ids that enter the Storm topology. Once a bolt processes a tuple, it sends an ACK to the acker bolt. While the tuples leave the topology, the acker bolt removes their identifiers. In case of failure, all tuples that have not received an acknowledgment will be re-processed.

Although Storm does not come with a machine learning library, it interfaces perfectly with the SAMOA tool offering implementations for classification and clustering algorithms for Big Data streams.

C. Spark

Spark is a project supported by the Apache community. It was initiated by the Berkeley University of California, it is a distributed data processing platform, written in Java and Scala. Fig. 4 shows the architecture of Spark cluster. Spark has four libraries running on top of the Spark engine: Spark SQL, MLlib for machine learning, GraphX, and Spark Streaming for stream processing.

Spark stores data in a distributed data set called resilient distributed dataset (RDD), which represents an immutable collection of read-only fault-tolerant objects (Python, Java, or Scala). Data will be stored in memory in a partitioned manner across multiple machines in the cluster. The sequence of operations to obtain a particular RDD is stored in the RDD itself, so in case of failures, the RDD can be rebuilt.

The operations that can be performed on RDDs are: The transformation which allows the construction of a new RDD from one or more input RDDs and the action which is an operation producing a single value or writing of the output on the disk.

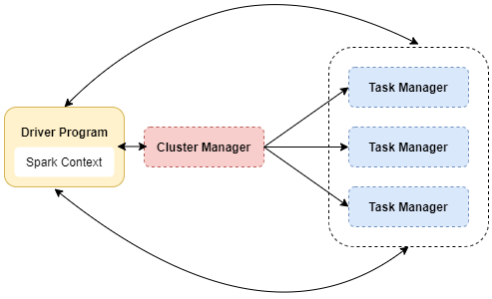

Fig. 4. Architecture of Spark Cluster.

The stream abstraction is called Discrete Stream (D-Stream) defined as a set of short, stateless, deterministic tasks. In Spark, streaming computation is treated as a series of deterministic batch computations on small time intervals [12]. In this approach, an incoming stream is packaged into sequences of small chunks of data, which can then be processed by a batch system [16]. While this may be adequate for many projects, it is not a true real-time system [14].

Table I summarizes the main features of stream processing systems. Flink and Storm perform real streaming while Spark is able to do micro-batching. The three systems present two different programming models: the compositional model which provides basic building blocks, such as spouts and bolts while the declarative model offers operators to generate functional code capable of automatically creating the topology.

TABLE I. DISTRIBUTED STREAM PROCESSING ENGINES

	Spark	**Flink**	**Storm**
Source Model	Open source	Open source	Open source
Architecture	Master-Slave	master-slave	master-slave
Coordination	Zookeeper	Zookeeper	Zookeeper
Execution model	Batch, micro-batch	Batch, streaming	Streaming
Stream abstraction	DStream	DataStream	Tuple
Supported languages	Java, Python, R, Scala	Java, Scala	Any
Associated ML tools	MLlib, Mahoot, H2O	FlinkML SAMOA	SAMOA
In-memory processing	yes	yes	yes
Low latency	<= Time of micro-batching	yes	yes
Fault tolerance	yes	yes	yes

IV. BIG DATA ARCHITECTURES BENCHMARKING

In this section, we will review the two main stream processing architectures, Kappa and Lambda. We will then compare these two architectures in order to propose the architecture which best suits our use case.

A. Lambda Architecture

Lambda architecture was introduced by Marz and Warren [17] and provides a set of architectural principles to ingest and process both stream and batch data, in a single Big Data architecture [18]. The main idea behind the Lambda architecture is to combine real-time and batch processing in a single technology stack. This ensures low latency and provide better results. Fig. 5 shows the main components of the architecture. In fact, it combines several paradigms into a single system that breaks down processing into three layers: Batch, serving, and speed. The batch layer is generally based on Hadoop technology, it stores the raw data as soon as it arrives and calculates the views that will be sent to the serving layer for indexing and tracking results. Batch processing is performed regularly at a defined interval on all data. The speed layer processes new data that are not already delivered in the batch view for rapid consumption. Incoming data are sent to both the batch layer and the speed layer for processing. When the system is queried, the results of the two layers will be merged to calculate the response. The views from the batch layer and those from the speed layer in near real-time are sent to the serving layer. This later indexes the views transmitted by the batch and speed layers so they can be queried in Ad-Hoc with low latency.

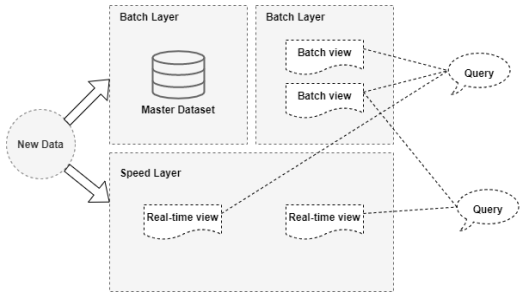

Fig. 5. Lambda Architecture.

B. Kappa Architecture

Unlike Lambda, Kappa is only focused on the processing of data streams. Described for the first time by Kreps [19], the key idea for its implementation is not to replace the Lambda architecture but to use a single layer in real-time to process both stream and batch data. Fig. 6 shows the two layers of the Kappa architecture: The streaming layer responsible for data processing and the serving layer for querying results. Only one code is used for data processing, unlike lambda where it is necessary to generate a code version for each layer. Similarly, queries only search in one location instead of two for Lambda (speed and batch views). The Kappa architecture is more suitable for cases where permanent data storage is not necessary.

Table II presents a brief comparison of the features of the Lambda and Kappa architectures.

The Lambda and Kappa architectures provide both real-time and historical data analysis in a single environment. Lambda uses two separate technology stacks to manage batch and stream processing. However, Kappa offers the possibility of building streaming and batch processing system based on the same technology. This is one major advantage of the Kappa architecture compared to the Lambda architecture. On the other hand, the Lambda architecture wins over Kappa when it comes to storing very large data sets (in terabyte range) and which can be processed more efficiently in Hadoop for large-scale historical analysis.

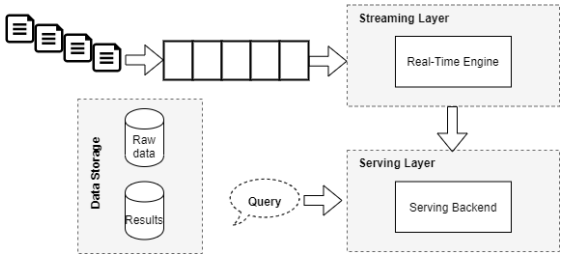

Fig. 6. Kappa Architecture.

TABLE II. COMPARISON OF THE DIFFERENT FEATURES OF LAMBDA AND KAPPA ARCHITECTURES

	Layers	Processing	Basic technologies	Scalability	Real-time
Lambda	Batch, serving and real-time layer	Batch and real-time	Immutable	Yes	Isn't accurate
Kappa	Stream processing and serving layer	Real-time	Immutable	Yes	Accurate

V. OUR PROPOSED BIG DATA ANALYTICS ARCHITECTURE FOR REAL-TIME STUDENT ATTENTION DETECTION

Traditional Computer Vision (CV) systems have limitations in terms of reliability, scalability and fault tolerance. In fact, in this type of system, the CV processing unit collects and processes the data at the same time. Thus, in case of failure, the data of the video stream will be lost and the processing will be interrupted. Detecting a node failure and switching the processing to another node will fragment the data and increase the error rate on the output result. In order to ensure a reliable and efficient data processing of a large-scale video stream, it is necessary to have a highly scalable, fault-tolerant and loosely coupled distributed system. Video stream analysis requires large-scale parallel processing, fast extraction of features from each frame, deployment of multiple machine-learning libraries, and returns processing results in different formats. In our system, we propose a Big Data architecture that best meets the requirements mentioned above. It will be based on the Kappa architecture principle for real-time processing of the video stream and will also allow permanent storage of this data. As shown in Fig. 8, it has five main layers, the data producer, data ingestion, flow processing, storage and presentation.

A. Design and Implementation

1) Data collection: The video stream is produced by a cluster of a high definition IP (HD 1080P: 1920 x 1080 (16/9) video at 30Fps) cameras designed to provide real-time video streams. The volume of data video is so important that it can easily reach the terabyte scale.

The cameras that make up the cluster provide data with precise specifications such as codec, resolution or the number of frames per second. For this, each camera has an ID that identifies it and keeps a mapping between the camera and its specifications. In this way, the stream processor can easily create images and video sequences from the video stream.

2) Data ingestion: For large scale stream acquisition, we choose to use Apache Kafka, that is, basically an open-source messaging system consisting of three components: Message Producer, Message Broker and Message Consumer. Kafka exchanges data between applications. Using the OpenCV library, we convert the video stream coming from cameras to images. Each image is stored in a JSON object. The content of this object consists of six fields: Id(ID of the camera), cols(number of columns), rows(number of rows in a frame), type (are OpenCV Mat-specific details), data(is a base-64 encoded string for the byte array of the frame), and the timestamp (the time at which the frame was generated). The JSON object obtained is a record which will constitute the body of a message to send to Kafka broker. Given the size of the frame that will be transported in the JSON objects, compression is needed to reduce the transfer time and ensure real-time processing. The compressed message is then transmitted to the producer, then to the Kafka Broker who is responsible for delivering it to consumers (stream processing unit).

3) Stream processing: Stream processing is built on Apache Storm, a distributed computing infrastructure for data stream processing and allows programming in python. Storm does not have an API to handle the low-level image processing. To overcome this, we use the OpenCV library which provides low-level image processing services on the top of Storm. As shown in Fig. 7, the stream processing unit consists of three main layers: The image pre-processing, mainly responsible for performing basic image processing tasks such as restoration (JSON to frame), encoding images/video, image enhancement, grayscale images, etc. The features extraction is responsible for extracting the features required to the analysis. These features will be provided as input to the third layer which is the distributed data mining. To implement this layer, we will use the Apache SAMOA a platform for mining Big Data streams [20]. APACHE SAMOA is both a framework and a library. As a framework, it allows the algorithm developer to abstract from the underlying execution engine, and therefore, reuse their code on different engines such as Storm, Flink, Samza, and Apex [21]. As a library, Apache SAMOA contains implementations of state-of-the-art algorithms for distributed machine learning on streams. For classification, Apache SAMOA provides a Vertical Hoeffding Tree (VHT), a distributed streaming version of a decision tree. For clustering, it includes an algorithm based on CluStream. For regression, HAMR, a distributed implementation of Adaptive Model Rules. The library also includes meta-algorithms such as bagging and boosting [21][22].

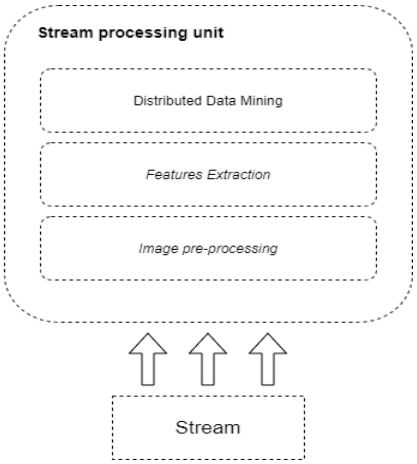

Fig. 7. The Stream Processing unit's Layers.

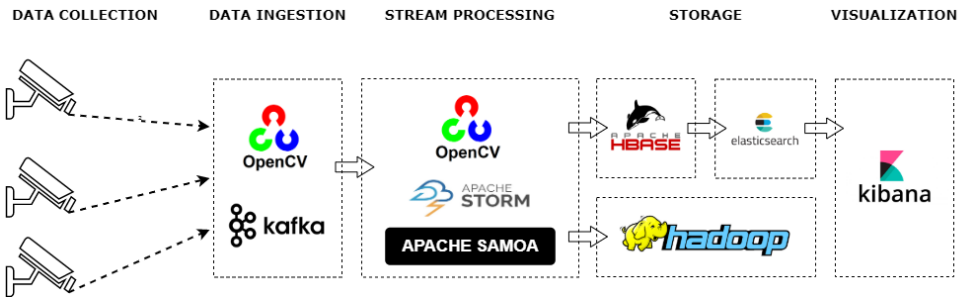

Fig. 8. Video Stream Analytics System Architecture.

4) Storage: The size of the video streams coming from multiple cameras installed in the different classrooms of a university can easily reach the Exabyte. So, scaling up, must be supported. Hence, storing such amounts of data is a real challenge. To solve this issue, video streams received from different cameras, through Kafka, encapsulated in JSON objects, are saved as MP4 files on the Hadoop Distributed File System storage (HDFS) of the Hadoop cluster. The storage will be distributed to ensure a fast flow recording. This also will ensure high data availability and the ability to perform distributed processing. Metadata for video streams and the result of video analysis are stored in an Hbase database, a data result duplication will be done in Elasticsearch for a possible visualization.

The received video streams are stored as 90 seconds video files. The size of a video file is strongly related to its length and it impacts the speed of storage, transmission, and analysis. The duration of a 90-second video file is chosen after taking into account the intra-cluster bandwidth, processing, machine performance, and fault tolerance. A small video file is transmitted faster than a large file. Furthermore, in case of failure, retransmission is faster.

The average size of a 90 second video stream is 16.2 GB, considering the average frame size which is 6M. Single-camera recording video (equivalent to 8 hours) requires a capacity of 5,184 TB. Table III summarizes the storage capacity required to store video stream for different durations.

5) Presentation: Visualization of analysis results is provided by Kibana, the official GUI visualization platform for Elasticsearch. Kibana has 4 main components: Discover, visualize, dashboards, and management. It provides access to the data stored in the Elasticsearch cluster through a Web interface that supports search, visualizes and analyzes functionalities.

TABLE III. DISK SPACE REQUIREMENTS FOR ONE WEEK OF RECORDED VIDEO STREAM

Duration	Size of a frame	Size of a video
90 seconds	6MB	16,2 GB
5 minutes	6MB	54GB
1 Hour	6MB	648GB
1 Day	6MB	5,184 TB
1 Week	6MB	31,104TB

VI. DISCUSSION

We have drawn up a comparison of the two most known architectures in Section IV. We discuss here the criteria for choosing our architecture and to what extent it meets the requirements of our problem.

The purpose of this study is to set up a real-time Big Data architecture capable of responding to the constraints of detecting and monitoring the student's attention at large scale. In other words, it must handle and process continuously and in real-time a large stream of images coming from the HD cameras placed in the classrooms. The video stream must also be stored for later analysis, to serve as a support for creating attention detection datasets or for training and validation tasks.

Our architecture is inspired by the concept of Kappa architecture since we only involve the processing of image data streams. The high capabilities of Apache Kafka make it possible to handle large incoming data streams. To overcome the real-time processing problem, Storm is used to pre-process the image streams, as well as the features extraction tasks. This is guaranteed, as Storm can be plugged-in as a pre-processor of the stream for each Kafka topic. In order to achieve real-time processing, the Storm topology must be created inside of it. For this purpose, we have coupled the use of Storm with SAMOA to benefit from its distributed machine learning libraries. The business logic of our architecture implies the need to store the data received for possible uses. Storage is provided by Hadoop with the possibility of processing which gives our architecture a hybrid form and makes it benefit from the advantages of the two traditional architectures.

The constraints regarding scalability and reliability were respected. Our architecture can scale-out on-demand to adapt to the increased load resulting from an expansion of university establishments and the number of students. Storm and Kafka clusters can scale horizontally. In addition, in case of machine failures, the data replication mechanisms allow fault tolerance, without loss of data.

VII. CONCLUSION

The main concern of this paper was to study the different technologies and architectures of Big data to propose an architecture capable of dealing with the problems linked to the high scale transition for student's attention detection systems. We compared the two main architectures on the market, namely, Lambda and Kappa, in order to choose the one that best meets our needs. Given the constraint of real-time

processing, we chose to base our architecture on the Kappa architecture which is simple and requires only a single implementation of the business logic. The choice of Apache Storm was not arbitrary but deduced from a comparison of stream processing tools. Storm is real-time processing with no restrictions on the implementation language. Furthermore, to implement machine learning algorithms in a distributed way, Storm can interface perfectly with Apache SAMOA.

The main limitation of this study is the lack of real-world data, which we take into consideration in future research. The next step of our work is to create a data set for the student's attention detection and to analyze it with several machine learning algorithms. Finally, we intend to perform the test and performance evaluation of our architecture.

REFERENCES

[1] J. Whitehill, Z. Serpell, Y.-C. Lin, A. Foster, and J. R. Movellan, "The faces of engagement: Automatic recognition of student engagementfrom facial expressions," IEEE Trans. Affect. Comput., vol. 5, no. 1, pp. 86–98, 2014.

[2] Krithika L.B and Lakshmi Priya GG, "Student Emotion Recognition System (SERS) for e-learning Improvement Based on Learner Concentration Metric," Procedia Comput. Sci., vol. 85, pp. 767–776, 2016.

[3] J. Zaletelj and A. Košir, "Predicting students' attention in the classroom from Kinect facial and body features," EURASIP J. Image Video Process., vol. 2017, no. 1, p. 80, Dec. 2017.

[4] N. Vettivel, N. Jeyaratnam, V. Ravindran, S. Sumathipala, and S. Amarakecrthi, "System for Detecting Student Attention Pertaining and Alerting," in 2018 3rd International Conference on Information Technology Research (ICITR), 2018, pp. 1–6.

[5] P. Goldberg et al., "Attentive or Not? Toward a Machine Learning Approach to Assessing Students' Visible Engagement in Classroom Instruction," Educ. Psychol. Rev., Dec. 2019.

[6] J. Gantz and D. Reinsel, "Extracting value from chaos," IDC iview, vol. 1142, no. 2011, pp. 1–12, 2011.

[7] S. Ali, A. Rauf, and J. Ahmad, "Protecting Unauthorized Big Data Analysis using Attribute (Data) Relationship."

[8] G. Glossary, "Big Data defintion." 2013.

[9] P. Zikopoulos, C. Eaton, and others, Understanding big data: Analytics for enterprise class hadoop and streaming data. McGraw-Hill Osborne Media, 2011.

[10] S. Madden, "From databases to big data," IEEE Internet Comput., vol. 16, no. 3, pp. 4–6, 2012.

[11] S. Sagiroglu and D. Sinanc, "Big data: A review," in 2013 international conference on collaboration technologies and systems (CTS), 2013, pp. 42–47.

[12] M. A. Lopez, A. G. P. Lobato, and O. C. M. B. Duarte, "A Performance Comparison of Open-Source Stream Processing Platforms," in 2016 IEEE Global Communications Conference (GLOBECOM), 2016, pp. 1–6.

[13] A. Fundation, "storm.apache.org," 1394. [Online]. Available: https://storm.apache.org.

[14] S. Landset, T. M. Khoshgoftaar, A. N. Richter, and T. Hasanin, "A survey of open source tools for machine learning with big data in the Hadoop ecosystem," J. Big Data, vol. 2, no. 1, p. 24, 2015.

[15] I. Bartolini and M. Patella, "Real-Time Stream Processing in Social Networks with RAM3S," Futur. Internet, vol. 11, no. 12, p. 249, 2019.

[16] S. Shahrivari, "Beyond batch processing: towards real-time and streaming big data," Computers, vol. 3, no. 4, pp. 117–129, 2014.

[17] N. Marz and J. Warren, Big Data: Principles and best practices of scalable real-time data systems. New York; Manning Publications Co., 2015.

[18] V. Persico, A. Pescapé, A. Picariello, and G. Sperl\'\i, "Benchmarking big data architectures for social networks data processing using public cloud platforms," Futur. Gener. Comput. Syst., vol. 89, pp. 98–109, 2018.

[19] J. Kreps, "Questioning the lambda architecture," Online Artic. July, p. 205, 2014.

[20] G. De Francisci Morales, "SAMOA: A platform for mining big data streams," in Proceedings of the 22nd International Conference on World Wide Web, 2013, pp. 777–778.

[21] N. Kourtellis, G. D. F. Morales, and A. Bifet, "Large-scale learning from data streams with Apache SAMOA," in Learning from Data Streams in Evolving Environments, Springer, 2019, pp. 177–207.

[22] T. Vasiloudis, F. Beligianni, and G. De Francisci Morales, "BoostVHT: Boosting distributed streaming decision trees," in Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, 2017, pp. 899–908.