A Deep Intelligence Framework for Online Video Processing

Weishan Zhang[1], Liang Xu[1], Zhongwei Li[1], Qinghua Lu[1], Yan Liu[2]
[1]College of Computer and Communication Engineering, China University of Petroleum, Qingdao, China
[2]Electric and Computer Engineering, Concordia University, Canada
zhangws@upc.edu.cn

Abstract—**Video data has become the largest source of big data. Due to challenges of video data complexities, velocity, and volume of accumulated videos, efficient and intelligent video processing at run time is required for public security and other surveillance applications. In this paper, we propose an intelligent online video processing framework that can help to address these challenges. This framework combines two cloud computing technologies, namely stream processing and batch processing from Storm and Hadoop respectively. Besides, in order to mine deep knowledge from videos, deep learning techniques are utilized to realize deep intelligence that can help to obtain knowledge hidden in video data. We have implemented the framework by combining different architecture styles including publish-subscribe, shared data, MapReduce, and layered architecture. Evaluations on performance, fault-tolerance, and scalability show the effectiveness of the proposed framework.**

Keywords—cloud computing, deep learning, fault tolerance, video processing, framework

I. INTRODUCTION AND MOTIVATION

Video data are accumulatively collected everyday due to wide deployment of cameras in applications such as smart transportation and security surveillance. It was pointed that surveillance videos are the biggest source of big data [1]. Making full use of these surveillance videos may play critical roles for security reasons, where fast, reliable, and intelligent processing of videos is of utmost importance. Due to the intrinsic complexities, we expect that the video processing can be finished online in a seconds scale.

The research on analyzing videos has been explored extensively like various feature extraction, video summary, and so on. Recently, video and image processing research is deeply influenced by the success of deep learning initiated by Hinton [2]. Hinton obtained the world's best classification result for the ImageNet problem with DCNN (Deep Convolutional Neural Network) [3], where none artificial features are used. DCNN has recently been applied to problems including face detection, activity recognition, and achieved unprecedented good results. We call the knowledge obtained from big data using deep learning as 'deep knowledge' or ‘deep intelligence’, which can potentially achieve better analyzing results than other approaches.

Due to the volume, velocity, and complexities of video data, there has been some research on video processing using the latest computing technologies, such as Cloud Computing, in the flavor of offline processing style like Hadoop[1], or stream processing style like Storm[2]. For example, Intel proposed a solution to address large-scale, real-time video analytics architecture [4]. However, this work did not discuss how a specific stream processing technology such as Storm can be used for video processing in details, and deep learning was not used for video processing.

In a summary, the existing work falls short in engaging Cloud Computing technologies with deep learning for real time continuous video processing. Therefore, in this paper, we present such a deep intelligence video data processing framework that integrates offline cloud processing, online stream processing, and deep learning at run time. We first present key quality attributes for this framework and architecture styles applied in this framework. We then show an overall architecture of the framework. After that we evaluate the proposed framework for its performance, fault-tolerance, and scalability, with the discussions of the efficiency, our experiences of integrating different architecture styles, and threats to validity.

II. RELATED WORK

The work by Intel [4] showed that real-time response of video processing is possible, however the details of how Storm can be used. There is a Split & Merge architecture proposed in [5] for high performance video processing, which is a generalization of the MapReduce paradigm. This work did not address real time response for video processing. Similar ideas were also discussed in [6] where Hadoop is used to process videos, with similar drawbacks.

The most intriguing work for real time data processing is the general Lambda architecture proposed by Nathan Marz [7]. The Lambda Architecture solves the problem of computing arbitrary functions on arbitrary data at real-time by decomposing the problem into three layers: the batch layer, the serving layer, and the speed layer. The Lambda architecture serves as a base for intelligent video processing where fast video processing become possible by using Storm.

There are some efforts to understand big video and image data using deep learning. For example, time-space Convolutional Neural Network was used to conduct action recognition [8]. However, the existing efforts did not take full advantage of cloud computing for their parallel, distribution, storage and other advantages, in order to obtain deep knowledge at run time. Although there is work to make use of

[1] http://hadoop.apache.org/

[2] http://storm.apache.org/

MapReduce to enhance machine learning techniques [9], their application to continuous video processing and run time performance evaluation were not the focus.

III. Architecture of the Deep intelligence framework

We adopt a quality driven approach [10] for the architecture design of the deep intelligence framework. Based on the Lambda architecture [7] and our previous work [11], we identified the following key quality attributes for the framework:

- Performance. There are two types of performance requirements: the batch processing for large volume of historical video data, e.g. conducting a video summary, which is not performance critical; the second one is for real time processing situations like emergency cases, which is performance critical.
- Availability. The framework should be reliable to allow hardware failures. It also needs to provide video processing capabilities to meet certain quality of service requirements.
- Scalability. This means that the scale of the "cloud" can be dynamically extended, and with the support of adding new video sensing devices and processing nodes dynamically.
- Modifiability. As big data processing techniques are evolving quickly, the framework should allow easy changes of its functionalities, so that future modifications can be easily conducted.
- Portability and usability. The framework needs also to provide a unified way to manage video equipment and video data. At the same time, it should support different Linux style of operating systems.

Actually there are also other very important quality attributes, for example safety, and testability, which are not the focus of this paper.

Considering these quality attributes, a number of software architecture styles [10] are chosen for the design and implementation of the framework:

- Service oriented architecture (SOA). Due to complexities of video data processing, and various technologies involved, it is important that the design of the framework following a service oriented approach, so that modifiability can be achieved, and also the upgrade of a functionality may not affect end users. This architecture style can also help on achieving portability.
- Publish-Subscribe. In order to decouple event consumers and providers, the framework shall use a publish/subscribe event mechanism in order to handle events on monitoring framework running status. The Publish-Subscribe style helps to achieve availability of the framework in the way that mal-function states can be detected, and then corresponding operations can be initiated.
- MapReduce. As the framework needs to analyze enormous volumes of video data, the MapReduce style can efficiently process distributed large data set in parallel. The MapReduce can help to achieve performance and availability requirements.
- Shared Data: Video data will be used by different processing components, for example, background subtraction, video summary, and so on, therefore Shared Data pattern is used to achieve sharing of video data. The Shared Data style can help on performance attributes.
- Layered architecture. Functionalities of the framework may evolve separately, and the processing of big video data can also be separated into different types and may evolve independently. Therefore, a layered architecture style is used to achieve separation of different concerns. This architecture style can help on modifiability, portability and usability.

Considering these quality attributes and the chosen architecture styles, we design the architecture of the framework as shown in Figure 1, together with main data types exchanged between layers. This architecture is in general a layered architecture. We have also shown main software packages deployed in different layers.

The bottom layer is the Data Retrieval layer. This layer is a generalization of video data collection, where a video streaming server and a WebCam component are used to receive different type of video data from various cameras.

The second layer is Data Processing layer, which is actually composed by two sub-packages, namely the package of Offline Processing based on Apache Hadoop, and the other package of Online Processing based on Apache Storm stream processing. Considering the nature of different video processing tasks, some features like video summary, video encoding/decoding for large amount of video data are within the scope of offline processing where no real time requirements are needed. Some important deep learning offline training components are located in this package that includes DCNN training component, DBN (Deep Belief Network) training component.

Correspondingly, some lightweight video processing tasks are assigned to the Online Processing package, for example background subtraction. In this package, the recognition at real time is achieved based on the training results from the offline neural network training, including the recognition of events, objects and behaviors.

The top most layer is the Domain Service layer, which can be used to develop different domain applications, like smart transportation, smart campus, and so on. This layer is actually an aggregation of the offline processing results and online processing results. For example, to know the traffic condition for a certain road, it processes the historical data using the offline processing capabilities, and also process the video collected at run time using the online processing features.

The key classes for online processing are shown in Figure 2. This class diagram shows The *ForegroundObjectBolt* is a Storm bolt that removes background; the *PrioriKnowledgeFilterBolt* is used to locate the moving targets and filter out the object of classification according to the prior knowledge; finally the *ClassificationBolt* is used to classify the moving targets.

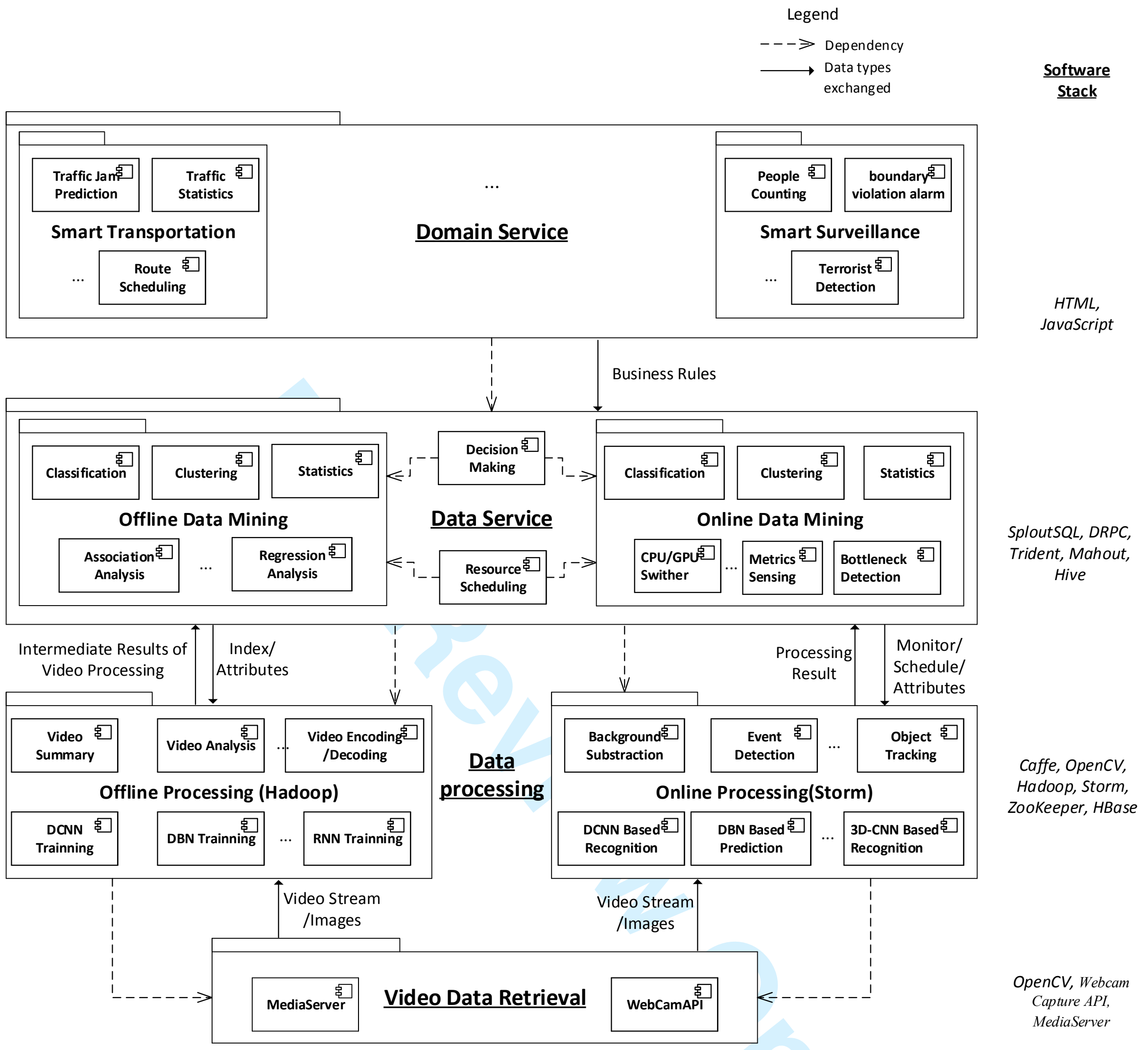

Figure 1. Architecture of the Deep Intelligence Framework and deployed software stack.

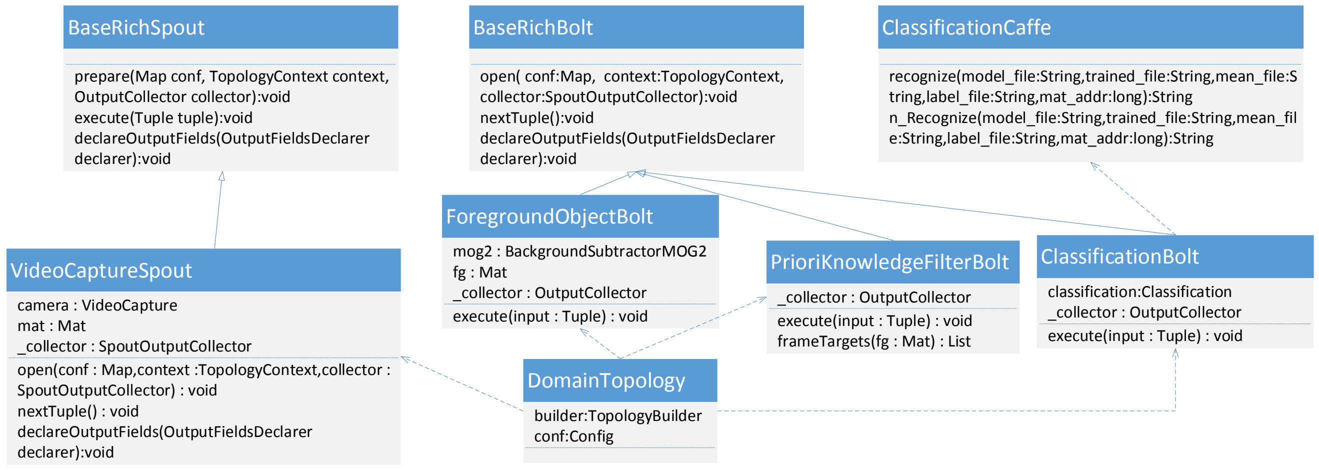
BaseRichSpout
prepare(Map conf, TopologyContext context, OutputCollector collector):void
execute(Tuple tuple):void
declareOutputFields(OutputFieldsDeclarer declarer):void
BaseRichBolt
open(conf:Map, context:TopologyContext, collector:SpoutOutputCollector):void
nextTuple():void
declareOutputFields(OutputFieldsDeclarer declarer):void
ClassificationCaffe
recognize(model_file:String,trained_file:String,mean_file:String,label_file:String,mat_addr:long):String
n_Recognize(model_file:String,trained_file:String,mean_file:String,label_file:String,mat_addr:long):String
ForegroundObjectBolt
mog2 : BackgroundSubtractorMOG2
fg : Mat
_collector : OutputCollector
execute(input : Tuple) : void
PrioriKnowledgeFilterBolt
_collector : OutputCollector
execute(input : Tuple) : void
frameTargets(fg : Mat) : List
ClassificationBolt
classification:Classification
_collector : OutputCollector
execute(input : Tuple) : void
VideoCaptureSpout
camera : VideoCapture
mat : Mat
_collector : SpoutOutputCollector
open(conf : Map,context :TopologyContext,collector : SpoutOutputCollector) : void
nextTuple() : void
declareOutputFields(OutputFieldsDeclarer declarer):void
DomainTopology
builder:TopologyBuilder
conf:Config

Figure 2. Main class diagram of online processing components

IV. Evaluating the Framework

In order to validate the proposed framework for its usability in video data real time intelligent processing, we have conducted a number of evaluations in terms of performance, fault-tolerance, and scalability. A use case of traffic statistics application (TSA) based on DCNN is used to count the number of cars, motors and pedestrians that pass across cameras.

In this paper, we mainly focus on real-time layer performance of using DCNN since the performance requirement is permanent. To evaluate the performance, we first design various DCNNs, and train these DCNNs with the Offline Processing components in order to get an appropriate network that has sufficient accuracy, and also meet real time requirement for object recognition. Then we design and deploy Storm Topology using the obtained DCNN parameters, and test the processing time for a single frame. After this, a client from the Online Data Mining layer will use Trident DRPC to obtain real-time views.

For testing scalability, multiple cameras are simulated by reading video files from local video streams. The testing of fault-tolerance is accomplished by killing some nodes in Storm.

A. Experiement Setup

There are 9 IBM3650 servers (64G RAM, 24 Cores, 12T Storage) deployed as a cloud infrastructure. Each node is running Ubuntu 14.04 server, and the modified deep learning tool Caffe [12] is deployed on every node. The configuration of the test bed is shown in Table 1, where the role is denoted as:

a: Namenode, b: DataNode, c: supervisor, d: Nimbus, e: ZooKeeper, f: DRPC Server, g: SploutSQL, h: ResourceManager, i: NodeManager, j: Web server.

Table 1. Configuration of the experiment cloud infrastructure

Node	1	2	3	4	5	6	7	8	9
Roles	a, d, f, h	b, c, f, i	b, c, f, i	b, c, f, i	b, c, f, g, i	b, c, f, i	b, c, e, f, i	b, c, e, f, i	b, c, e, f, i, j

The Hadoop Cluster is composed of mainly two parts. The first part is HDFS (Hadoop Distribute File System) that consists of Namenode and Datanode. zthe second part is MapReduce, a large-scale distributed computing paradigm based on the Divide-Conquer principle. The MapReduce framework consists of a single master *ResourceManager* and one slave *NodeManager* per cluster node.

The Storm cluster nodes are divided into two categories: the Master Nodes and the Work Nodes. The Master Node running on a Nimbus daemon, which is responsible for allocating resources and assigning tasks to worker nodes in the Storm cluster, and by receiving the worker nodes heartbeat information to monitor the running status of the cluster.

Zookeeper is running on node 7, 8 and 9 to coordinate Nimbus and Supervisor. Node 5 is a Splout SQL node and node 9 is also a Web server.

B. Performance Evaluation

We create a dataset that includes car, motor and person on the offline processing layer. There are 46906 training images and 8274 validation images in total. We use various networks to train on the dataset and partial results are shown in Table 2.

Table 2. Performance of the training of the DCNNs

Name	Image size	Layers	Training time	recognition time	Accuracy
Net 1	32*32	14	1h 1m30s	6.82ms	92.36%
Net 2	224*224	20	4h 57m43s	959.76 ms	95.85%

The size of the image and the network affect training and recognition speed, and recognition accuracy at the same time. From Table 2, we can see that the larger a picture and the more complex a network, the higher recognition rate we can get. However, the processing time for classification (recognition) is also getting longer. Considering the accuracy and real-time performance trade-off, Net1 is a practical choice.

This well-trained Net1 is then used for classification in the online processing layer by a specific Storm Bolt. We measure the background subtraction, target localization, and classification using ForegroundObjectBolt, PrioriKnowledgeFilterBolt and VehicleClassificationBolt (classification Bolt) as shown in Table 3.

Table 3. The processing time for a single frame (ms).

processing	ForegroundObjectBolt	PrioriKnowledgeFilterBolt	VehicleClassificationBolt
1	10.724	4.069	7.862
2	11.541	3.754	8.0
3	11.118	3.980	7.510
Average	11.13	3.93	7.79

TSA then uses Trident DRPC to obtain real time views from Storm. The latency for small queries as in our case takes 10ms in average. More intense DRPC queries can take longer of course, we can solve this problem by allocating more resources.

In the Domain Service layer, data services are integrated to adapt to different application domains. For example to predict the traffic condition based on the results from the Data Service Layer on how many cars since 9 o'clock, then we can know the traffic condition. In this case, it only takes around 10ms to generate the synthesis of results from both the Offline Processing and Online Processing.

Overall, it takes about 43 ms (11.33+3.93+7.79+10+10) after a video frame's acquisition to its target being recognized. TSA can achieve real-time performance in the second scale.

C. Scalability

The real time video stream of a camera is simulated by reading video files from a local disk. This way of simulation can eliminate the effect of network bandwidth in order to solely consider the scalability. Then we test the TSA's scalability by changing the number of cameras. There is a new topology job[3] submitted to Storm to deal with real-time video from each newly added camera. All these topologies are executed in parallel.

As shown in Figure 3, the performance of the test bed scales as more cameras are added. The limit of our test bed is to handle at least 100 cameras at real time. When it reach to 150 cameras, it is getting worse that the background subtraction takes around 50 ms that is beyond real time requirements.

[3] https://storm.apache.org/documentation/Tutorial.html

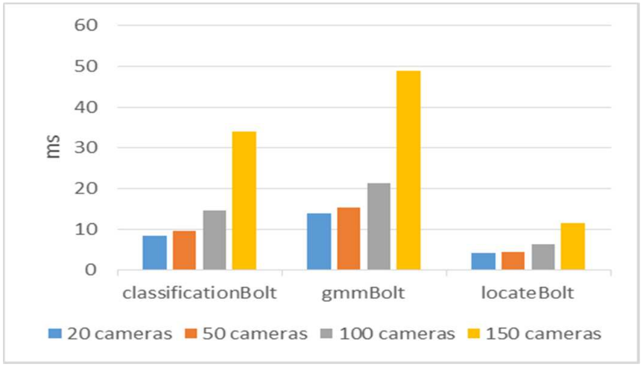

Figure 3. Scalability with more cameras

D. Fault tolerance

When workers die, Storm will automatically restart these workers in a few seconds. If a node dies, the workers on this node will be restarted on other nodes like nothing happened. We tested how this affects video processing by killing some nodes in storm. In case 1, we used seven supervisors (kill node 9) to handle eight cameras' data, and six supervisors (kill node 8,9) for case 2 and only four supervisors (kill node 6,7,8,9) for case 3. The processing time of each frame in different cases is shown in Figure 4, where the processing time of a frame keeps relatively stable, which shows that the proposed framework is fault-tolerant.

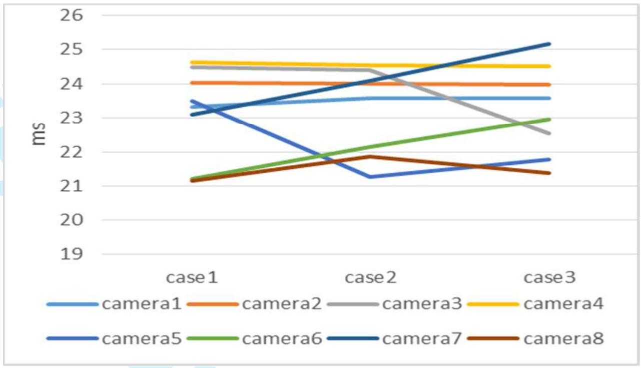

Figure 4. Fault tolerance when killing nodes

E. Discussion

From these evaluations, we can see that the proposed framework has good potentials for scalability, when adding new video sensing devices can be handled without much effects on the performance. For the fault tolerance, the framework can elegantly handle node failures where the processing time of each frame keeps stable. Most importantly, performance evaluations show that it takes only around 43ms to perform background subtraction, and recognition of the targets in a video frame, which is acceptable for real time object recognition.

We have set up the test bed with open source software packages, and normal server clusters deployed with our framework. The testing environment has been used for other evaluations including face recognition, object tracking, which also shows that the proposed framework is effective and efficient for continuous real time video processing.

A number of architectural styles are integrated in our framework. The Layered architecture style helps to achieve separation of different data processing concerns. The SOA is enacted with clearly designed interface contracts and makes components in different layers loosely coupled. Shared Data pattern is used to achieve sharing of video data between different processing components that span multiple layers, implemented using SOA. MapReduce is used as means to achieve parallel processing of video data, and is the one for actual implementations of some algorithms. The Publish-Subscribe pattern is used to decouple Storm generated events with event consumers that monitor the cluster status in order to do an optimized scheduling. The challenges of integrating SOA with layered architecture style is how to harmoniously interact different features in different layers through well-defined interface contracts, and how to use Shared Data to provide effective data partition for MapReduce.

References

[1] Tiejun Huang. Surveillance Video: The Biggest Big Data. Computing Now, vol. 7, no. 2, Feb. 2014, IEEE Computer Society.

[2] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. Science, 313(5786):504–507, 2006

[3] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[C]//Advances in neural information processing systems. 2012: 1097-1105.

[4] Hu, Xiao, Zhihong Yu, Huan Zhou, Hongbo Lv, Zhipeng Jiang, and Xiang Zhou. "An Adaptive Solution for Large-Scale, Cross-Video and Real-Time Visual Analytics." In Multimedia Big Data (BigMM), 2015 IEEE International Conference on, pp. 212-215. IEEE, 2015.

[5] Pereira, Rafael, Marcello Azambuja, Karin Breitman, and Markus Endler. "An architecture for distributed high performance video processing in the cloud." In Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, pp. 482-489. IEEE, 2010.

[6] Tan, Hanlin, and Lidong Chen. "An approach for fast and parallel video processing on Apache Hadoop clusters." In Multimedia and Expo (ICME), 2014 IEEE International Conference on, pp. 1-6. IEEE, 2014.

[7] N. Marz and J. Warren. Big Data: Principles and best practices of scalable realtime data systems. Manning Publications, 2013.

[8] Shuiwang Ji,Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 35(1):221–231, 2013.

[9] Bekkerman, Ron, Mikhail Bilenko, and John Langford, eds. Scaling up machine learning: Parallel and distributed approaches. Cambridge University Press, 2011.

[10] L. Bass, P. Clements, R. Kazman, Software architecture in practice, Addison-Wesley, 2012

[11] Weishan Zhang and Klaus Marius Hansen and Mad Ingstrup. A Hybrid Approach to Self-management in a Pervasive Service Middleware. Knowledge-Based Systems. 67: 143-161 (2014)

[12] Jia, Yangqing and Shelhamer, Evan and Donahue, Jeff and Karayev, Sergey and Long, Jonathan and Girshick, Ross and Guadarrama, Sergio and Darrell, Trevor. Caffe: Convolutional Architecture for Fast Feature Embedding. arXiv preprint arXiv:1408.5093 (2014)