A Deep-Intelligence Framework for Online Video Processing

Weishan Zhang, Liang Xu, Zhongwei Li, and Qinghua Lu,
China University of Petroleum

Yan Liu, Concordia University, Montreal

// A video-processing framework combines stream processing, batch processing, and deep learning to realize deep intelligence that can help reveal knowledge hidden in video data. Evaluations of performance, scalability, and fault tolerance showed the framework's effectiveness. //

APPLICATIONS SUCH AS SMART transportation and security surveillance collect vast amounts of video data daily. Surveillance videos are the biggest source of big data.[1] Making full use of these videos might be critical for security applications, which require fast, reliable, and intelligent video processing. Owing to this task's intrinsic complexities, we expect the online processing of such video to occur on a scale of seconds.

Video analysis research has extensively explored such topics as feature extraction and video summary. Recently, video- and image-processing research has been deeply influenced by the success of deep learning initiated by Geoffrey Hinton and Ruslan Salakhutdinov.[2] They obtained the world's best classification result for the ImageNet problem with a *deep convolutional neural network* (DCNN), which uses no artificial features.[3] Researchers have recently applied DCNNs to problems including face detection and activity recognition and achieved unprecedented good results. We call the knowledge obtained from big data using deep learning "deep knowledge" or "deep intelligence," which potentially can achieve better analysis results than other approaches.

Owing to the volume, velocity, and complexities of video data, some video-processing research has employed the latest computing technologies, such as cloud computing, using offline processing or stream processing. (For more on some of this research, see the sidebar.) However, that research has fallen short in engaging cloud-computing technologies with deep learning for real-time continuous video processing. To meet that need, we developed a framework that integrates offline cloud processing, online stream processing, and runtime deep learning. Evaluations of the framework have shown its effectiveness.

Our Framework's Architecture

To design our framework, we adopted a quality-driven approach.[4] On the basis of the Lambda Architecture (see the sidebar) and our previous research,[5] we identified five key quality attributes:

- *Performance*. There are two types of performance requirements. The first is batch processing of large volumes of historical video data (for example, conducting a video summary), which is not performance critical. The second type is for

0740-7459/16/$33.00 © 2016 IEEE

RELATED WORK IN VIDEO PROCESSING

Intel researchers showed that real-time video processing is possible.[1] However, they didn't discuss in detail how to use a specific stream-processing technology such as Storm for video processing, and they didn't use deep learning for video processing. Rafael Pereira and his colleagues proposed the Split&Merge architecture for high-performance video processing,[2] which is a generalization of MapReduce. They didn't address real-time video processing. Hanlin Tan and Lidong Chen discussed similar ideas; they used Hadoop to process videos, with similar drawbacks.[3]

The most intriguing approach to real-time data processing is Nathan Marz and James Warren's Lambda Architecture.[4] It computes arbitrary functions on arbitrary data in real time by decomposing the problem into a batch layer, a serving layer, and a speed layer. It's the basis for fast, intelligent video processing using Storm.

Some researchers have tried to use deep learning to understand big video and image data. For example, Ji Shuiwang and his colleagues used a time–space convolutional neural network to recognize actions.[5] However, such research hasn't fully exploited cloud computing for parallelism, distribution, storage, and other advantages to obtain deep knowledge at runtime. Although researchers have used MapReduce to enhance machine-learning techniques, they didn't focus on continuous video processing and runtime performance evaluation.[6]

References

1. X. Hu et al., "An Adaptive Solution for Large-Scale, Cross-Video and Real-Time Visual Analytics," *Proc. 2015 IEEE Int'l Conf. Multimedia Big Data* (BigMM), 2015, pp. 212–215.
2. R. Pereira et al., "An Architecture for Distributed High Performance Video Processing in the Cloud," *Proc. 2010 IEEE Int'l Conf. Cloud Computing*, 2010, pp. 482–489.
3. H. Tan and L. Chen, "An Approach for Fast and Parallel Video Processing on Apache Hadoop Clusters," *Proc. 2014 IEEE Int'l Conf. Multimedia and Expo* (ICME 14), 2014, pp. 1–6.
4. N. Marz and J. Warren, *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*, Manning Publications, 2013.
5. J. Shuiwang et al., "3D Convolutional Neural Networks for Human Action Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, 2013, pp. 221–231.
6. R. Bekkerman, M. Bilenko, and J. Langford, eds., *Scaling Up Machine Learning: Parallel and Distributed Approaches*, Cambridge Univ. Press, 2011.

real-time processing situations (for example, emergency cases), which is performance critical.
- *Availability*. The framework must be reliable to allow hardware failures. It must provide dependable services to users.
- *Scalability*. The cloud must be able to scale up, with the ability to add video-sensing devices and processing nodes dynamically.
- *Modifiability*. Because big data processing techniques are evolving quickly, the framework must allow easy changes of its functionalities so that future modifications are easy.
- *Portability and usability*. The framework must also provide a unified way to manage video equipment and video data. It must also support different Linux-style OSs.

(Other important quality attributes exist—for example, safety and testability—but aren't part of this article's focus.)

Considering the five quality attributes, we chose the following software architecture styles[4] for our framework:

- *Service-oriented architecture* (SOA). Owing to the complexities of video data processing and the various technologies involved, the framework must employ an SOA. This will enable modifiability, and functionality upgrades won't affect users. An SOA can also help achieve portability.
- *Publish–subscribe*. To decouple event consumers and providers, a publish–subscribe mechanism should handle events while monitoring the framework's running status. This will help achieve the framework's availability such that users can detect malfunction states and initiate appropriate responses.
- *MapReduce*. The framework must analyze enormous volumes of video data. MapReduce is a large-scale

TABLE 1

Mapping architecture styles to quality attributes.

Architecture style	Supported quality attributes
Service-oriented architecture	Portability, modifiability
Publish–subscribe	Availability, modifiability
MapReduce	Performance, scalability, availability
Shared Data	Performance, availability
Layered architecture	Modifiability, portability, usability

distributed-computing paradigm based on the divide-and-conquer strategy. It can efficiently process distributed large datasets in parallel. So, it can help meet performance, scalability, and availability requirements.

- *Shared Data*. Different processing components—for example, background subtraction and video summary—will use video data. So, the framework uses the Shared Data pattern to achieve sharing of video data, which can help improve performance.
- *Layered architecture*. Framework functionalities might evolve separately, and the processing of big video data can also be separated into different types and might evolve independently. So, the framework uses a layered architecture to separate the different concerns. This can support modifiability, portability, and usability.

Table 1 shows a mapping of the architecture styles to the quality attributes.

Figure 1 shows the architecture, the main data types exchanged between layers, and the main software packages in each layer. The bottom layer is the data retrieval layer. It's a generalization of video data collection, in which a media server and webcam API receive different types of video data from various cameras.

Next, the data-processing layer comprises the offline-processing package, based on Apache Hadoop (http://hadoop.apache.org), and the online-processing package, based on Apache Storm stream processing (http://storm.apache.org). Some video-processing tasks—for example, video summary and encoding and decoding of large amounts of video data—occur during offline processing, which involves no real-time requirements. The offline-processing package contains important deep-learning components such as the DCNN training and DBN (deep belief network) training components.

The online-processing package handles some lightweight video-processing tasks—for example, background subtraction. On the basis of the offline neural-network training results, this package achieves real-time recognition, including the recognition of events, objects, and behaviors.

Figure 2 shows the key classes for online processing. Three classes, all Storm bolts, were of particular interest in our study. `ForegroundObjectBolt` removes the background. `PrioriKnowledgeFilterBolt` locates the moving targets and filters out the object of classification according to the prior knowledge. `ClassificationBolt` classifies the moving targets.

The top layer is the domain service layer, which developers can use to create different domain applications such as smart transportation and smart campuses. This layer aggregates the offline- and online-processing results. For example, to determine a road's traffic conditions, it uses offline processing of the historical data and online processing of the runtime-collected video.

Evaluating the Framework

We evaluated our framework's performance, scalability, and fault tolerance. Our use case was a DCNN-based traffic statistics application (TSA) that counted the number of cars, motorcycles, and pedestrians that cameras detected.

The Setup

We used nine IBM 3650 servers (64 Gbytes of RAM, 24 cores, and 12 Tbytes of storage) deployed as a cloud. Each node ran the Ubuntu 14.04 server and a modified version of the Caffe deep-learning tool.[6] Table 2 shows the testbed configuration.

The Hadoop cluster mainly comprised two parts. The first part was HDFS (Hadoop Distribute File System), which consisted of a NameNode and DataNode. The second part was MapReduce, which consisted of a ResourceManager per cluster and a NodeManager per cluster node.

The Storm cluster nodes comprised a master node and worker nodes. The master node ran on a Nimbus daemon. It allocated resources, assigned tasks to worker nodes, and received the worker nodes' heartbeat information to monitor the cluster's running status.

ZooKeeper ran on nodes 7, 8, and 9 to coordinate the Nimbus and the Supervisor. Node 5 was a

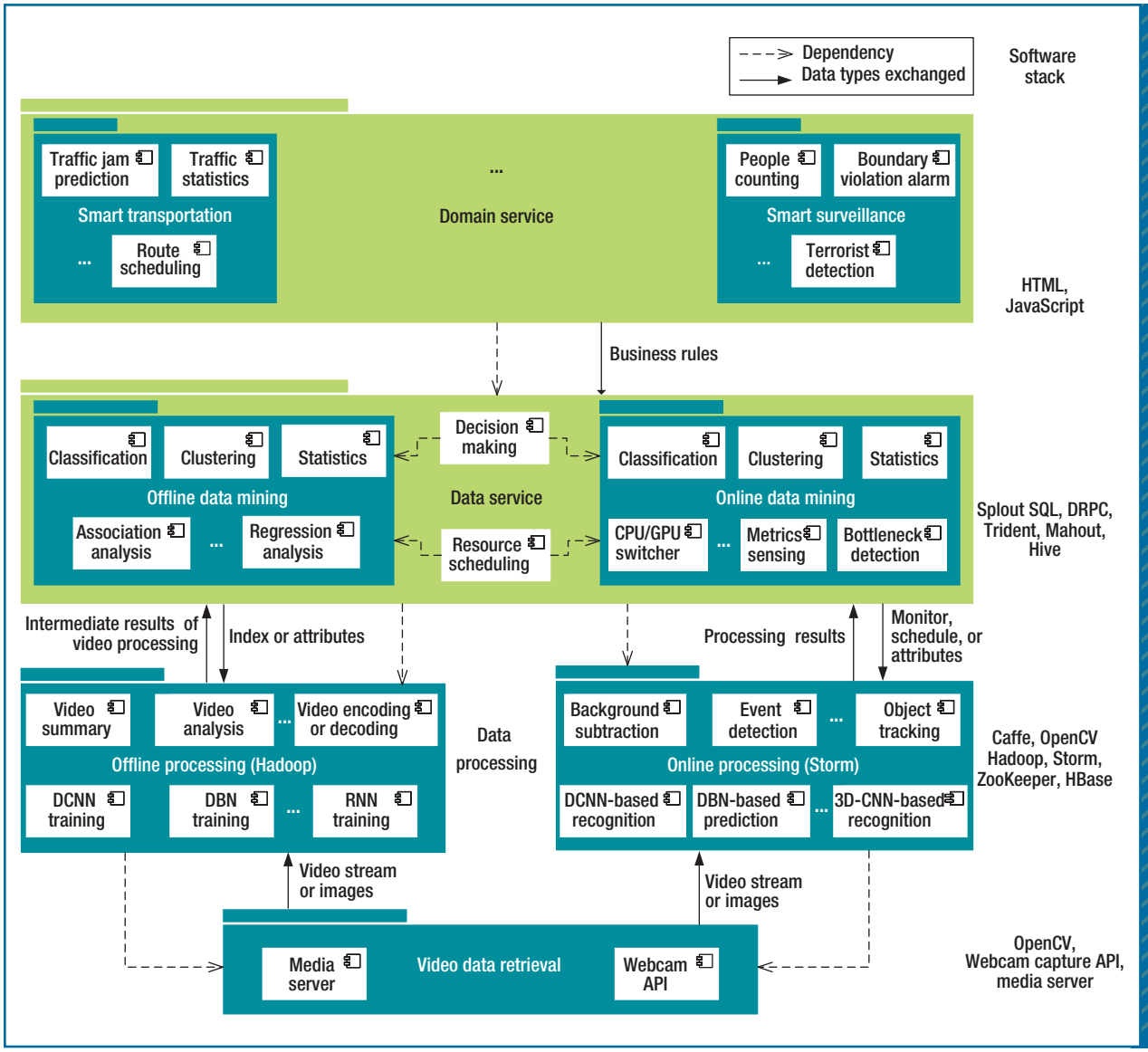

FIGURE 1. The architecture of the deep-intelligence framework and the deployed software stack. DCNN stands for deep convolutional neural network, DBN stands for deep belief network, and RNN stands for recurrent neural network.

TABLE 2

The configuration of the experiment's cloud infrastructure.*

	Node								
	1	2	3	4	5	6	7	8	9
Roles	Name-Node Nimbus DRPC server Resource-Manager	DataNode Supervisor DRPC server Node-Manager	DataNode Supervisor DRPC server Node-Manager	DataNode Supervisor DRPC server Node-Manager	DataNode Supervisor DRPC server Splout SQL Node-Manager	DataNode Supervisor DRPC server Node-Manager	DataNode Supervisor ZooKeeper DRPC server Node-Manager	DataNode Supervisor ZooKeeper DRPC server Node-Manager	DataNode Supervisor ZooKeeper DRPC server Node-Manager Webserver

** DRPC stands for Distributed Remote Procedure Call.*

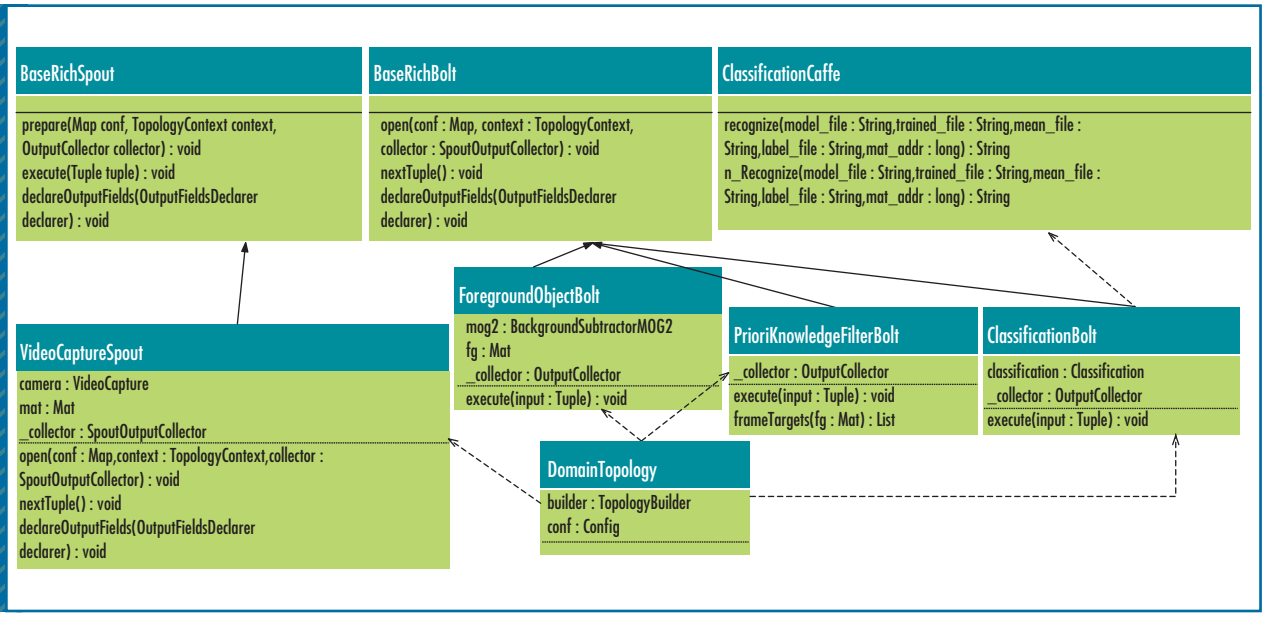

FIGURE 2. The main class diagram of the online-processing components. Three classes, all Storm bolts, were of particular interest in our study. **ForegroundObjectBolt** removes the background. **PrioriKnowledgeFilterBolt** locates the moving targets and filters out the object of classification according to the prior knowledge. **ClassificationBolt** classifies the moving targets.

TABLE 3

DCNN (deep convolutional neural network) training performance.

Network	Image size (pixels)	No. of layers	Training time	Recognition time (ms)	Accuracy (%)
1	32 × 32	14	1 h, 1 min, 30 s	6.82	92.36
2	224 × 224	20	4 h, 57 min, 43 s	959.76	95.85

Splout SQL node; node 9 was also a webserver.

Performance

In this article, we focus mainly on the real-time layer's performance using DCNNs because the performance requirement is permanent. For our performance evaluation, we created a dataset that included images of cars, motorcycles, and pedestrians on the offline-processing layer. The dataset contained 46,906 training images and 8,274 validation images. We then designed various DCNNs and trained them on the data to get a network that had sufficient accuracy and met the requirements for real-time object recognition.

Table 3 shows the results for two networks. As the image size grew and the network became more complex, the recognition rate increased. However, the training and recognition time also grew. Considering the accuracy and real-time-performance trade-off, Network 1 was a practical choice.

We then used the well-trained Network 1 for classification in the online-processing layer. We measured the background subtraction, target localization, and classification using **ForegroundObjectBolt**, **PrioriKnowledgeFilterBolt**, and **VehicleClassificationBolt**. Table 4 shows the performance for three consecutive runs of the online processing.

The TSA then used the Trident DRPC (Distributed Remote Procedure Call) to obtain real-time views from Storm. The latency for small queries was 10 ms on average. Of course, more-intense DRPC queries could take longer; we could solve this problem by allocating more resources.

TABLE 4

The processing time for one frame (ms), for three consecutive runs of the online processing.

Run	Online-processing class*		
	ForegroundObjectBolt	PrioriKnowledgeFilterBolt	VehicleClassificationBolt
1	10.724	4.069	7.862
2	11.541	3.754	8.000
3	11.118	3.980	7.510
Average	11.130	3.930	7.790

*ForegroundObjectBolt *removed the background.* PrioriKnowledgeFilterBolt *located the moving targets and filtered out the object of classification according to the prior knowledge.* VehicleClassificationBolt *classified the moving targets.*

In the domain service layer, data services are integrated to adapt to different application domains. For example, we could predict the traffic condition on the basis of the results from the data service layer regarding how many cars had passed the cameras since 9 o'clock. In this case, synthesizing the results of both offline and online processing took only approximately 10 ms each.

Overall, it took approximately 43 ms (11.13 + 3.93 + 7.79 + 10.00 + 10.00) from video frame acquisition to target recognition. So, the TSA could achieve real-time performance on the scale of seconds, as we would expect.

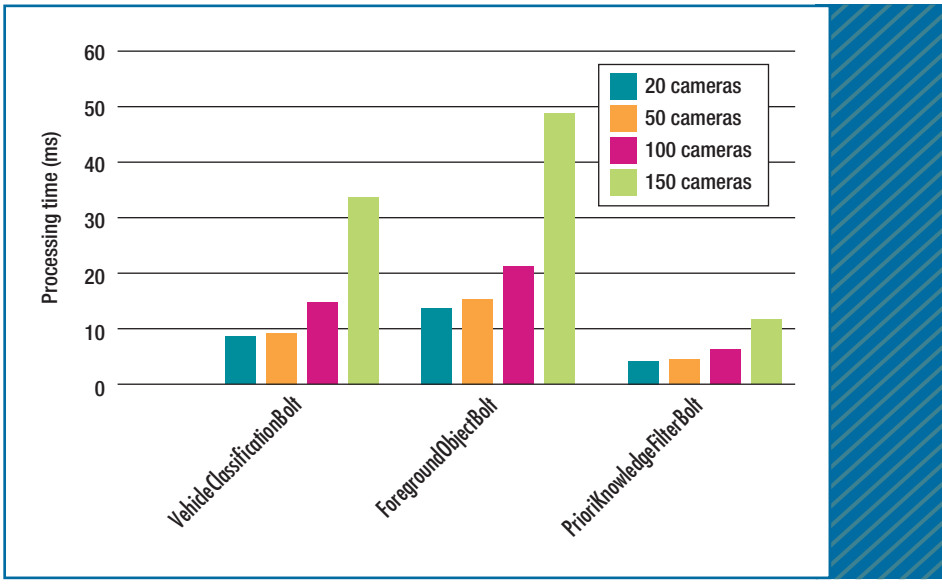

FIGURE 3. Scalability with more cameras. The testbed could handle at least 100 cameras in real time.

Scalability

We simulated a camera's real-time video stream by reading video files from a local disk. Such simulation can eliminate the effect of network bandwidth to consider only the scalability. Then, we tested the TSA's scalability by changing the number of cameras. The framework submitted a new topology job (https://storm.apache.org/documentation/Tutorial.html) to Storm to deal with real-time video from each newly added camera. These topologies were executed in parallel.

As Figure 3 shows, the testbed's performance scaled as we added cameras. It could handle at least 100 cameras in real time. However, when it reached 150 cameras, background subtraction took approximately 50 ms, which could hardly meet the real-time requirements.

Fault Tolerance

When workers die, Storm automatically restarts them in a few seconds. If a node dies, the workers on it restart on other nodes as if nothing happened.

To test how this affected video processing, we killed nodes in Storm. We evaluated three cases, each involving eight cameras. Case 1 used seven supervisors (we killed node 9). Case 2 used six supervisors (we killed nodes 8 and 9). Case 3 used only four supervisors (we killed nodes 6, 7, 8, and 9). Figure 4 shows

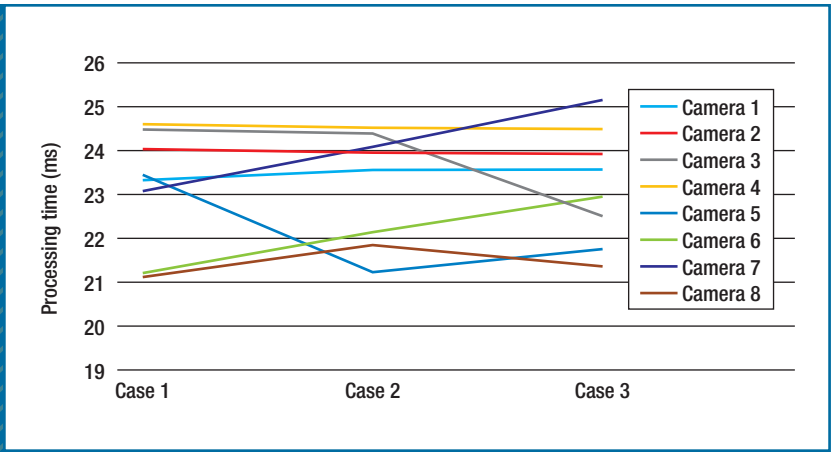

FIGURE 4. Fault tolerance when killing nodes. The processing time kept relatively stable, showing that our framework was fault tolerant.

each frame's processing time for the three cases. The processing time kept relatively stable, showing that our framework was fault tolerant.

We set up the testbed with open source software packages and deployed normal server clusters with our framework. We've used the testing environment for other evaluations, including face recognition and object tracking. These evaluations also showed that our framework was effective and efficient for continuous real-time video processing.

We feel that our framework has successfully integrated the five architecture styles. The SOA is enacted with clearly designed interface contracts and loosely couples components in different layers. Publish–subscribe decouples Storm-generated events and event consumers that monitor the cluster status to optimize scheduling. MapReduce achieves parallel processing of video data, and we've employed it for implementations of some algorithms. The Shared Data pattern shares video data among processing components spanning multiple layers, implemented with the SOA. Finally, the layered architecture helps separate different data-processing concerns. Integrating the SOA in a layered architecture posed two challenges: how to enable different features to interact harmoniously in different layers through well-defined interface contracts, and how to use shared data to provide effective data partitioning for MapReduce.

To make our framework more versatile, we're integrating more video processing and mining algorithms—for example, for shadow elimination and video event detection. We're also applying the framework to industry use, including smart oil fields (for example, oil production monitoring) and smart transportation for Qingdao City, China.

Acknowledgments

The research is supported by the Innovative Method special project of the Ministry of Science and Technology (grant 2015IM010300), National Natural Science Foundation of China (grant 61402533), and Natural Science Foundation of Shandong Province (grant ZR2014FM038), Key Technologies Development Plan of Qingdao Technical Economic Development Area. Weishan Zhang has been supported by the startup funds for Academic Top-Notch Professors in China University of Petroleum.

References

1. T. Huang, "Surveillance Video: The Biggest Big Data," *Computing Now*, vol. 7, no. 2, 2014; www.computer.org/web/computingnow/archive/february2014.
2. G.E. Hinton and R.R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," *Science*, vol. 313, no. 5786, 2006, pp. 504–507.
3. A. Krizhevsky, I. Sutskever, and G.E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Proc. Advances in Neural Information Processing Systems* (NIPS 12), 2012, pp. 1097–1105.
4. L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Addison-Wesley, 2012.
5. W. Zhang, K.M. Hansen, and M. Ingstrup, "A Hybrid Approach to Self-Management in a Pervasive Service Middleware," *Knowledge-Based Systems*, Sept. 2014, pp. 143–161.
6. Y. Jia et al., "Caffe: Convolutional Architecture for Fast Feature Embedding," 2014; http://arxiv.org/abs/1408.5093.

ABOUT THE AUTHORS

WEISHAN ZHANG is a full professor and deputy head of research in the China University of Petroleum's Department of Software Engineering. His research interests are big data platforms, pervasive cloud computing, and service-oriented computing. Zhang received a PhD in mechanical manufacturing and automation from Northwestern Polytechnical University. Contact him at zhangws@upc.edu.cn.

LIANG XU is a master's student in the China University of Petroleum's Department of Software Engineering. His research interests are big data processing, software architecture, and cloud computing. Xu received a bachelor of software engineering from the China University of Petroleum. Contact him at xuliang.upc.edu@gmail.com.

ZHONGWEI LI is an associate professor in the China University of Petroleum's College of Computer and Communication Engineering. His research interests include big data for petroleum engineering, algorithms, and social networks. Li received a PhD in petroleum engineering from the China University of Petroleum. Contact him at lizhongwei@upc.edu.cn.

QINGHUA LU is a lecturer in the China University of Petroleum's Department of Software Engineering. Her research interests include software architecture, cloud-computing dependability, and service engineering. Lu received a PhD in software engineering from the University of New South Wales. Contact her at dr.qinghua.lu@gmail.com.

YAN LIU is an associate professor in the Electrical and Computer Engineering Department at Concordia University, Montreal. Her research interests are data stream processing, cloud software architecture, distributed computing, software performance engineering, and adaptive systems. Liu received a PhD in computer science from the University of Sydney. She's a member of IEEE. Contact her at yan.liu@concordia.ca.