大規模データ処理フレームワークのストリーミング機能を利用した機械学習処理の評価

一瀬　絢衣† 　竹房あつ子†† 　中田　秀基††† 　小口　正人†

† お茶の水女子大学　〒 112-8610 東京都文京区大塚 2-1-1
†† 国立情報学研究所　〒 101-8430 東京都千代田区一ツ橋 2-1-2
††† 産業技術総合研究所　〒 305-8560 茨城県つくば市梅園 1-1-1
E-mail: †ayae@ogl.is.ocha.ac.jp, ††takefusa@nii.ac.jp , †††hide-nakada@aist.go.jp, ††††oguchi@is.ocha.ac.jp

あらまし　各種センサの普及やクラウドコンピューティング技術の習熟に伴い，お年寄りや子供のための安全サービスなどを目的としたライフログの利用が普及してきている．しかし，動画像解析のようなデータ量，計算量の多い処理をクラウドでリアルタイムに行うことは困難である．また，近年ディープラーニング技術の発達で，その高い精度から画像や音声の認識などに広く用いられているが，計算負荷が高いことが問題の一つとなっている．
本研究では，大規模データ処理のための高速かつ汎用性の高いエンジン Apache Spark のストリーミング機能を利用して，ディープラーニングフレームワークの一つである Chainer の機械学習処理を行い，動画像データ解析処理の高速化を図る．本稿ではクライアント側，クラウド側として 2 つの端末間で Apache Kafka を用いてストリーミング処理を行い，クラウド側で機械学習の識別処理を行った．2 つの端末間のネットワーク帯域を変化させ，1 秒あたりに識別処理を行える画像数，1 マイクロバッチの処理にかかる時間を計測し，性能を評価した．結果から，クラウド側の処理が全体のボトルネックとなっていること，低帯域環境においてリアルタイム処理を行うことが可能であることが示された．

キーワード　ストリーミング処理，Apache Spark，機械学習，リアルタイム処理

Evaluation of Machine Learning Processing Using a Streaming Function of a Large-scale Data Processing Framework

Ayae ICHINOSE†, Atsuko TAKEFUSA††, Hidemoto NAKADA†††, and Masato OGUCHI†

† Ochanomizu University　2-1-1 Otsuka, Bunkyouku Tokyo 112-8610 JAPAN
†† National Institute of Informatcs　2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430 JAPAN
††† National Institute of Advanced Industrial Science and Technology (AIST) 1-1-1 Umezono, Tsukuba, Ibaraki 305-8568 JAPAN
E-mail: †ayae@ogl.is.ocha.ac.jp, ††takefusa@nii.ac.jp , †††hide-nakada@aist.go.jp, ††††oguchi@is.ocha.ac.jp

1. はじめに

各種センサの普及やクラウドコンピューティング技術の習熟に伴い，お年寄りや子供のための安全サービスなどを目的としたライフログの利用が普及してきている．このようなサービスでは，一般家庭にサーバやストレージを設置して全ての処理を行うことは困難であるため，センサデータをそのまま送信してクラウドで処理するのが一般的である．センサデータの解析をクラウドで行う研究は多くなされており，Twitter のセンチメント分析など小規模かつ大量のデータをクラウド内で効率よく解析する手法が提案されている．しかし，動画像解析は連続的に大容量データを転送する必要があり，計算量，データ量の多い処理をクラウドでリアルタイムに行うことは困難である．
画像や映像の解析にはディープラーニング技術が広く使われている．ディープラーニングはニューラルネットワークの中で識別を行う中間層を多層化したものを用いた機械学習であり，精度やスピードの向上という点で注目されている．Chainer [1] や Caffe [2]，TensorFlow [3] といったディープラーニングのフレームワークも多く利用されているが，計算負荷が高いことが課題の一つとなっている．

本研究では，大規模データ処理のための高速かつ汎用性の高いエンジン Apache Spark(以降，Spark と呼ぶ) [4] のストリーミング機能を利用して，ディープラーニングフレームワークの一つである Chainer を用いて機械学習処理を行い，動画像データ解析処理の高速化を図る．本稿ではクライアント側，クラウド側として 2 つの端末間で Apache Kafka(以降，Kafka と呼ぶ) [5] を用いてストリーミング処理を行い，クラウド側で機械学習の識別処理を行った．様々な利用環境を想定して 2 つの端末間のネットワーク帯域を変化させ，1 秒あたりに識別処理を行える画像数を計測し，性能を調査した．実験から，クラウド側の処理が全体のボトルネックとなっていること，低帯域環境においてリアルタイムに識別を行うことが可能であることが示された．

2. 関 連 技 術

2.1 Apache Spark

Spark は，カリフォルニア大学バークレー校で開発が開始された，大規模データの格納，処理を目的とした分散処理フレームワークである．同じく分散処理フレームワークである Apache Hadoop で用いられている，MapReduce と呼ばれるバッチ処理に特化した処理方法に対し，Spark はデータをメモリに保存することで入出力を高速化することにより，処理全体の実行速度の向上を図る．このアプローチにより，レスポンスの速さが必要とされる対話的処理や，データを繰り返し処理する機械学習処理などに適している．

Spark は複数のコンポーネントで構成されており，その一つにストリームデータを処理する Spark Streaming がある．Spark Streaming は，数秒から数分ほどの短い間隔で繰り返しバッチ処理を行うマイクロバッチ方式によりストリームデータ処理機能を提供する．Spark Streaming は Apache Kafka や Amazon Kinesis，Twitter からのデータ取得や MQTT のプロトコルにも対応しており，それぞれのデータストリームに合ったアプリケーションを容易に作成できる．

2.2 Apache Kafka

Kafka は，大容量データを高スループット/低レイテンシに収集，配信することを目的に開発されている分散メッセージングシステムである．複数のサーバ上でクラスタとして実行可能であり，Kafka クラスタはトピックと呼ばれるカテゴリにキー，値，タイムスタンプから構成されたレコードのストリームを格納する．メッセージングモデルには送信側がデータの転送を開始する Push 型と，受信側がデータのリクエストを送ることによりデータの転送が開始される Pull 型という 2 つの大きな概念が存在する．Kafka は図 1 に示す構造により，この 2 つのメッセージングモデルの概念を一般化している．Producer と Kafka Cluster 間は Push 型のメッセージングモデルであり，Kafka Cluster と Consumer 間は Pull 型のメッセージングモデルである．これにより，Pull 型のデータの処理の分割による処理のスケーラビリティと，Push 型の複数プロセスへのデータのブロードキャストの両方を可能にしている．Kafka は最大限のスピードでデータを消費することを目標としているため，

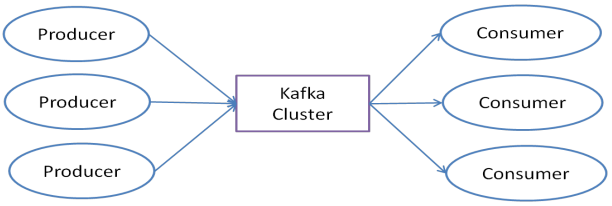

図 1　Apache Kafka

Consumer がデータの到着量を管理することにより Consumer の最大効率の達成を可能にしている．こういった技術から速度，耐久性に優れており，リアルタイムのデータパイプラインやストリーミングアプリケーションの構築に広く使われている．

2.3 Chainer

Chainer は，Preferred Networks が開発したディープラーニングフレームワークである．Python のライブラリとして提供されており，制御構造はすべて Python での記述が可能である．Chainer の特徴として，柔軟性，直感的，高速の 3 つが掲げられている．畳込みニューラルネットやリカレントニューラルネット，再帰型ニューラルネットなど様々なネットワークアーキテクチャをシンプルに実装でき，また，GPU にも対応している．他のフレームワークの多くが一度ニューラルネット全体の構造をメモリ上に展開し，その処理通りに順伝搬，逆伝播を実行するというアプローチであるのに対し，「Define-by-Run」方式と呼ばれる，ネットワーク構築と学習を同時に行う方式を採用しているのも大きな特徴である．動的にネットワークを定義するため手法の自由度が高く，複雑化していくディープラーニングの開発への対応が可能であると考えられている．また，インストールも容易にできることから，広く使われている．

3. 実　　験

本研究では，ストリーミング機能を用いた識別処理の性能を調査するため，クライアント，クラウド間のネットワーク帯域を変化させた環境下での，1 秒あたりに処理できる画像数を調査した．また，処理のリアルタイム性を調査するため，1 マイクロバッチの処理時間を計測した．

3.1 実 験 環 境

本研究では図 2 に示す構成で，クライアント側，クラウド側として 2 つの端末を用いてストリーム処理を行う．クライアント側は Kafka を用いてデータの送信を行う．クラウド側では Spark で Kafka からデータを受け取り，Python プログラムを実行して Chainer を呼び出す．ここで，Spark ではデータは RDD (Resilient Distributed Dataset) という，分散処理を前提としたオブジェクトのコレクションとして扱われるため，データは RDD として読み込み，RDD から Chainer の要求する型に変換する．マイクロバッチサイズを指定し，指定した時間ごとに区切られたデータに対し Chainer を呼び出して識別処理を行った．

実験には 0 から 9 の手書き数字の 28 × 28 画素の画像データに正解ラベルが与えられているデータセットである MNIST [6] を用いた．DataProducer から 500,000 枚のデータを流して画

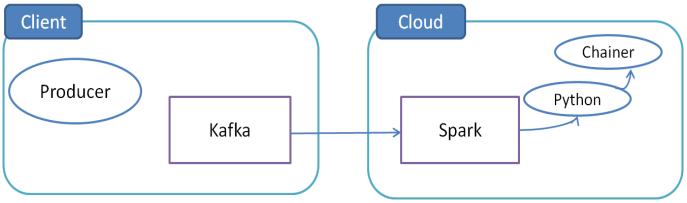

図 2　ストリーミング機械学習

表 1　実験で用いた計算機の性能

OS	Ubuntu 16.04LTS
CPU	Intel(R) Xeon(R) CPU W5590 @3.33GHz (8 コア) × 2 ソケット
Memory	8Gbyte

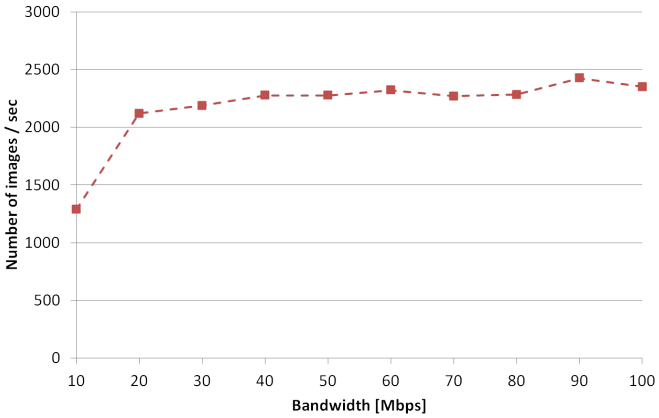

図 3　マイクロバッチサイズを 1 秒に指定した場合の 1 秒あたりの処理画像数

像の識別処理を行った．実験に用いた計算機の性能を表 1 に示す．クライアント側及びクラウド側で同質のノードを用い，その間のネットワーク帯域は 1Gbps となっている．ネットワーク帯域の制御には PSPacer [7] を用いた．

3.2　スループット計測

クライアント，クラウド間のネットワーク帯域を 10Mbps から 100Mbps まで変化させ，クラウド側でデータを受信し学習処理を行う時間と処理した画像数から，1 秒あたりの処理画像数を計測した．マイクロバッチサイズを 1 秒，2 秒，5 秒に設定して計測した結果をそれぞれ図 3，図 4，図 5，3 つの比較を図 6 に示す．横軸が端末間のネットワーク帯域を示し，縦軸が 1 秒あたりに処理できた画像の枚数を示す．全ての結果において 20Mbps ほどで処理性能が飽和し，ネットワーク帯域が広い場合においても 1 秒あたりの処理画像数は増加しないことが確認できた．つまり，1 秒あたりに送られる画像数が増加してもクラウド側の処理がボトルネックとなってしまい，全体のスループットが増加しないことがわかった．また，マイクロバッチサイズを 1 秒に設定した場合には 1 秒あたりの処理画像数が 2,300 枚ほどであるのに対し，5 秒に設定した場合には 2,500 枚ほどに増加している．これは，1 マイクロバッチサイズごとにデータをまとめて識別処理をしているため，Chainer の呼び出しオーバヘッドが削減されたためだと考えられる．

3.3　リアルタイム性評価

Spark Streaming のマイクロバッチサイズはデータサイズを

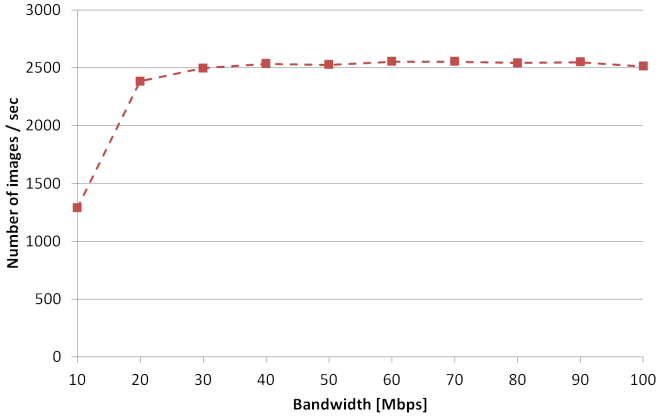

図 4　マイクロバッチサイズを 2 秒に指定した場合の 1 秒あたりの処理画像数

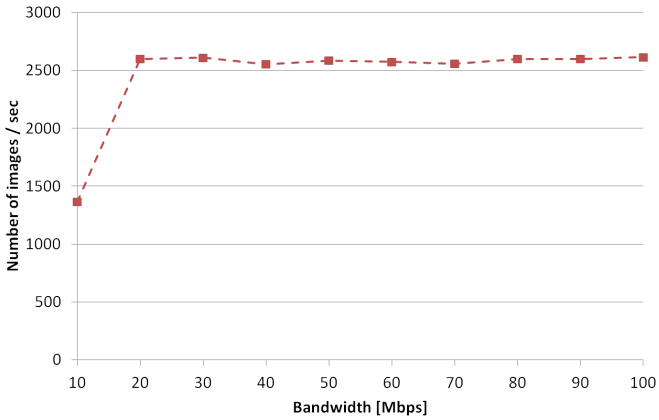

図 5　マイクロバッチサイズを 5 秒に指定した場合の 1 秒あたりの処理画像数

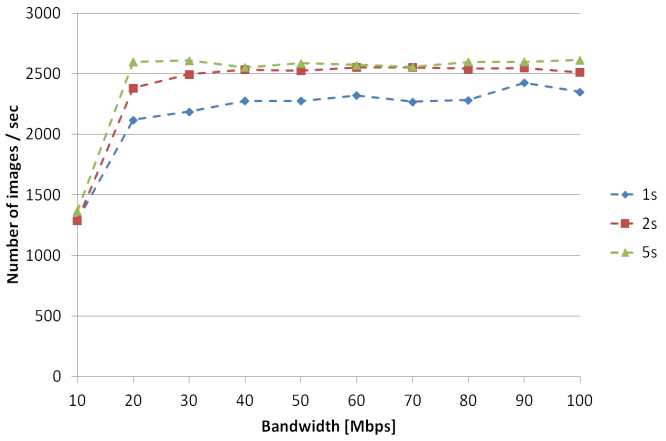

図 6　マイクロバッチサイズの変化による 1 秒あたりの処理画像数の比較

表しているため，1 マイクロバッチのデータ処理にかかる時間をタイムスタンプを用いて計測し，平均値を計算して 1 マイクロバッチの処理時間とした．マイクロバッチサイズを 1 秒，2 秒，5 秒に設定して計測した結果をそれぞれ図 7，図 8，図 9，3 つの比較を図 10 に示す．横軸が端末間のネットワーク帯域を示し，縦軸が 1 マイクロバッチの処理時間を示す．全ての場合において，10Mbps ではリアルタイムに処理が行われているが，20Mbps 以上になると帯域の変化に伴って処理時間も増加していることが確認できた．つまり，受信データが大きくなると処理が追い付かず，リアルタイムに処理できていないことが

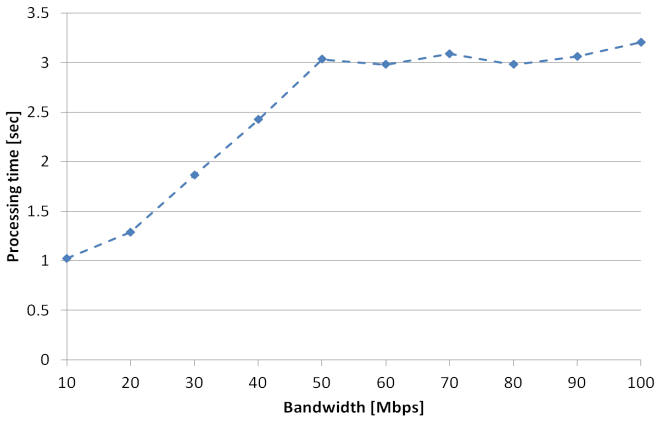

図 7　マイクロバッチサイズを 1 秒に指定した場合の 1 マイクロバッチの処理時間

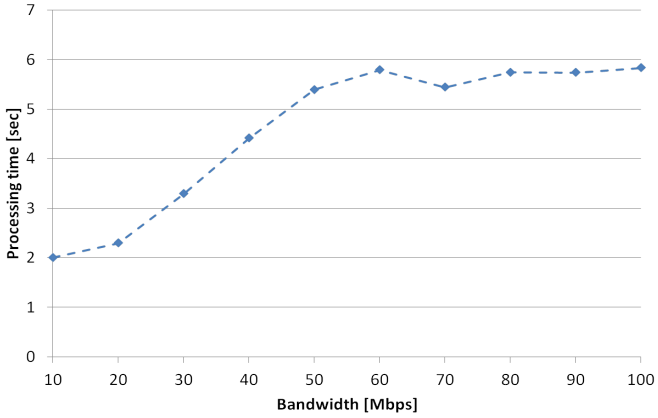

図 8　マイクロバッチサイズを 2 秒に指定した場合の 1 マイクロバッチの処理時間

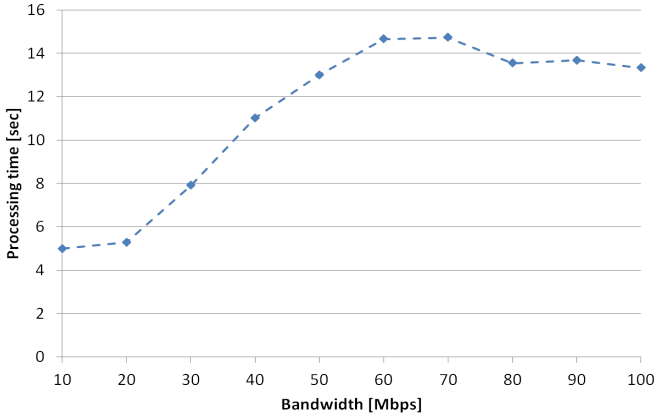

図 9　マイクロバッチサイズを 5 秒に指定した場合の 1 マイクロバッチの処理時間

わかった．Kafka においては Kafka Cluster と Consumer 間は Pull 型のメッセージシステムのデザインであるため，クラウド側の Chainer の機械学習処理のボトルネックから，データ転送が遅れていることが原因であると考えられる．また全ての結果で，60Mbps ほどで処理時間がマイクロバッチサイズの 3 倍ほどに達し，その後は横ばいになっていることが確認できた．これは Producer と Kafka Cluster 間のメッセージングシステムがボトルネックとなっているためだと考えられる．

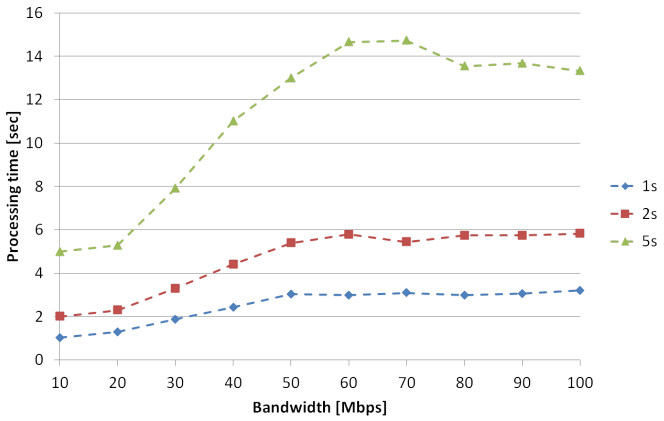

図 10　マイクロバッチサイズの変化による 1 マイクロバッチの処理時間の比較

4. 関 連 研 究

ディープラーニングを用いたストリームデータ解析は近年数多く研究されており，高速に処理するためのアルゴリズムや精度を向上させるアーキテクチャが検討されている [8] [9] [10]．しかし，これらはストリームデータを一つの計算機で実行することが前提とされている．本研究はセンサ・クラウド間のデータ送信を考慮した全体のスループットを考慮している点で異なる．また，Spark Streaming を利用してストリーム処理を行っているため，Spark の機能を利用した拡張を容易に行うことが可能である．

Spark Streaming は様々な技術に応用されており，Miucin ら [11] は Spark Streaming で実装された解析ツールを提供するツールキットである DINAMITE を提案している．DINAMITE の解析ツールは高度なデバッグ情報ですべてのメモリアクセスを計測し，プログラマがメモリのボトルネックを特定するのを支援する．Chen ら [12] は，現在広く使われているルールベースのシステムのスピード，スケーラビリティ，フォルトトレランスといった課題への対処として Spark Streaming を利用している．我々は Spark Streaming を用いた機械学習処理により，一般家庭のライフログ解析の効率化を図る．

5. まとめと今後の課題

本研究では大規模データ処理フレームワーク Spark のストリーミング機能を利用して Chainer を用いた機械学習処理を行う事により，リアルタイムセンサデータ解析処理における性能要件を検討した．実験から，低帯域環境でリアルタイムでの画像識別が可能であることが確認できた．高帯域環境では Kafka の構造からクラウド側の処理がボトルネックとなってしまっていることが確認できた．また，高帯域環境でのデータ転送では，Producer と Kafka Cluster 間の転送がボトルネックとなっていることが確認できた．

今後の課題としては，クラスタを用いた分散実行により全体のボトルネックとなっているクラウド側の処理の高速化を図る．また，Producer と Kafka Cluster 間のボトルネックとなっている原因を詳しく解析し，解決する．

謝　　辞

この成果の一部は，JSPS 科研費 JP16K00177 および国立研究開発法人新エネルギー・産業技術総合開発機構（ＮＥＤＯ）の委託業務の結果得られたものです．

文　　献

[1] Tokui, S., Oono, K., Hido, S. and Clayton, J.: Chainer: a Next-Generation Open Source Framework for Deep Learning, *In Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)* (2015). 6 pages.

[2] Jia, Y. et al.: Caffe: Convolutional Architecture for Fast Feature Embedding, *arXiv preprint arXiv:1408.5093* (2014).

[3] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems (2015). `http://download.tensorflow.org/paper/whitepaper2015.pdf`. pp. 1-19.

[4] Apache Spark, https://spark.apache.org/.

[5] Apache Kafka, https://kafka.apache.org/.

[6] Lecun, Y., Cortes, C. and Burges, C. J. The MNIST Database of handwritten digits, http://yann.lecun.com/exdb/mnist/.

[7] Takano, R., Kudoh, T., Kodama, Y. and Okazaki, F.: High-resolution Timer-based Packet Pacing Mechanism on Linux Operating System, *IEICE Transactions on Communications*, Vol. E94.B, No. 8, pp. 2199–2207 (2011).

[8] Qing, L., Zhaofan, Q., Ting, Y., Tao, M., Yong, R. and Jiebo, L.: Action Recognition by Learning Deep Multi-Granular Spatio-Temporal Video Representation, *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval*, ICMR '16, New York, NY, USA, ACM, pp. 159–166 (2016).

[9] Ye, H., Wu, Z., Zhao, R.-W., Wang, X., Jiang, Y.-G. and Xue, X.: Evaluating Two-Stream CNN for Video Classification, *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*, ICMR '15, New York, NY, USA, ACM, pp. 435–442 (2015).

[10] Read, J., Perez-Cruz, F. and Bifet, A.: Deep Learning in Partially-labeled Data Streams, *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, New York, NY, USA, ACM, pp. 954–959 (2015).

[11] Miucin, S., Brady, C. and Fedorova, A.: End-to-end Memory Behavior Profiling with DINAMITE, *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, New York, NY, USA, ACM, pp. 1042–1046 (2016).

[12] Chen, Y. and Bordbar, B.: DRESS: A Rule Engine on Spark for Event Stream Processing, *Proceedings of the 3rd IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*, BDCAT '16, New York, NY, USA, ACM, pp. 46–51 (2016).