Kafkaを利用したリアルタイム動画像解析フレームワークのレプリケーションによる性能変化の調査

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, oguchi@is.ocha.ac.jp, ††takefusa@nii.ac.jp , †††hide-nakada@aist.go.jp

あらまし　近年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が活用されている．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．
我々は，Kafka と Spark Streaming を用いた，複数カメラからの動画像収集とその解析処理を効率よく行う動画像解析フレームワークを構築している．本稿では，データ収集を行う Kafka のレプリケーションの設定による性能の変化に注目し，提案フレームワークのスループットを計測した．実験から，レプリケーション数を増やすことによりネットワークの輻輳が起き，データ転送のスループットが低下し，システム全体のスループットが低下することがわかった．また，ネットワーク帯域を十分に増やすことにより性能劣化を軽減でき，レプリケーション数を増やした場合にも Worker の処理能力に応じたスループットで解析処理できることが確認できた．
キーワード　ストリーミング処理，Apache Spark，Apache Kafka，リアルタイム処理

Performance Evaluation of a Kafk-based Real-time Video Analysis Framework Using a Replication Function

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, oguchi@is.ocha.ac.jp, ††takefusa@nii.ac.jp , †††hide-nakada@aist.go.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．
我々は，Kafka と Spark Streaming を用いた，複数カメラからの動画像収集とその解析処理を効率よく行うことを目的としたリアルタイム動画像解析フレームワークを構築している [4]．本稿では，データの収集を行う Kafka のレプリケーションの設定

による性能の変化に注目し，提案フレームワークのスループットを計測した．実験から，レプリケーション数を増やすことによりネットワークの輻輳が起き，データ転送のスループットが低下し，システム全体のスループットが低下することがわかった．また，ネットワーク帯域を十分に増やすことにより性能劣化を軽減でき，レプリケーション数を増やした場合にも Worker の処理能力に応じたスループットで解析処理できることが確認できた．

2. 動画像解析フレームワークの概要

本フレームワークで利用する Spark と Kafka について説明した後，提案するフレームワークの概要を述べる．

2.1 Apache Spark

Spark は，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 [5]．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 と呼ばれるファイルへの入出力を前提とした分散処理に対し，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．このアプローチは，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を行う．

2.2 Apache Kafka

Kafka は，大容量データを高スループット/低レイテンシで収集，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 [6]．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 型がある．Kafka は図 1 に示す構造のように，Producer，Broker (Kafka Cluster)，Consumer で構成され，Producer と Broker 間は Push 型のメッセージングモデル，Broker と Consumer 間は Pull 型のメッセージングモデルとなっている．トピックを生成する際にデータのパーティション数やレプリケーション数を設定することができ，分割されたデータが複数の Consumer に分配される．レプリケーションは，現状生存している Broker に対してパーティションごとに配置され，Broker の追加，削除に応じて自動で再配置される．同期レプリケーション，非同期レプリケーションの設定が可能であり，同期レプリケーションは全てのレプリカから応答が返ってから Consumer にデータが公開され，非同期レプリケーションは 1 つのレプリカから応答が返った時点でデータが公開される．デフォルトでは非同期レプリケーションに設定されている．

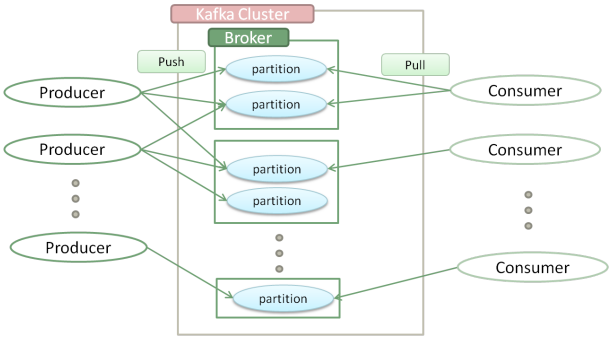

図 1 Apache Kafka

2.3 Chainer

Chainer は，Preferred Networks 社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．

2.4 リアルタイム動画像解析フレームワーク

提案するリアルタイム動画像解析フレームワークでは，図 2 に示す構成でストリーム処理を行う．クライアント側は，Kafka の Producer として動作し，データの転送を行う．クラウド側では Kafka の Broker がデータが保持し，Spark の Worker が Kafka の Consumer として動作する．Worker は Python プログラムを実行し，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，Worker はデータを RDD として読み込み，RDD から Chainer の要求する型に変換する．Spark のマイクロバッチ処理に用いられ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う．

3. スループット計測

Kafka のレプリケーション機能を用いることでデータの可用性を高めることができるが，レプリケーション処理による性能劣化が懸念される．本研究では，データ転送スループットと提案フレームワークを用いたデータの識別スループットを計測し，レプリケーションによる性能の変化を調査した．

3.1 実験概要

図 3 に示すクラスタ構成で，スループットを計測する．黄色

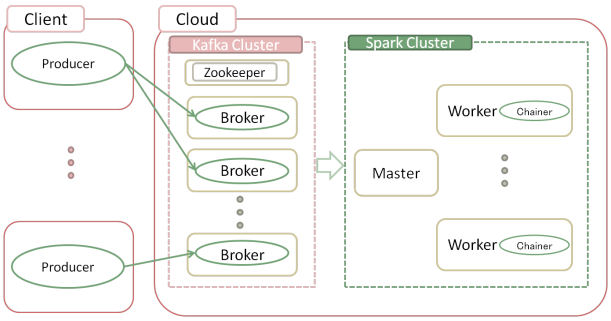

図 2　リアルタイム動画像解析フレームワークの概要

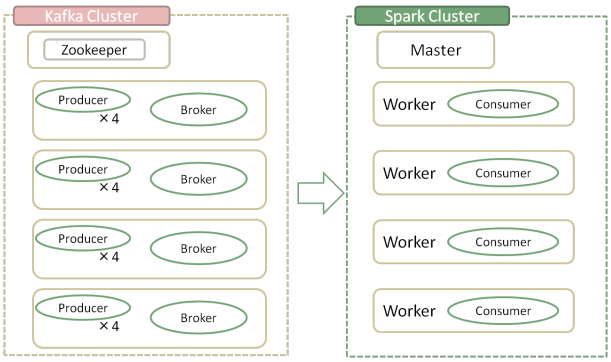

図 3　実 験 構 成

い四角が物理ノードを表す．クライアント側及びクラウド側で同質のノードを用いた (表 1)．Kafka クラスタ，Spark クラスタ共に 5 ノード用いており，それぞれクラスタの管理を行うノードが 1 台，データの処理を行うノードが 4 台である．端末間のネットワーク帯域は全てのノード間で一定とし，1Gbps と 10Gbps のそれぞれの場合で計測する．Producer は Broker と同じノード内で実行し，合計 16 個の Producer を動かす．それぞれの Producer で 10 万枚のデータを送信し，合計 160 万枚のデータを送信する．Producer 内で画像データをまとめて送信する画像数を Kafka のバッチサイズとし，1，5，10，20 で比較した．Spark Streaming のマイクロバッチサイズは 1 秒に固定し，Worker は 1 秒分のデータを Broker へ取りに行くことを繰り返す．識別処理を行う際には，Worker 内でデータを 100 枚にまとめ，100 枚ごとに Chainer を呼び出す．レプリケーション数を 1,2,3 とし，Producer 側で計測した全 Producer の合計送信スループット，Worker でデータの読み込みのみを行った場合の全 Worker の合計読み込みスループット，Producer からのデータ転送と同時に Worker でデータを読み込み Chainer のデータ識別処理を行った場合の全 Worker の合計データ処理スループットを計測した．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 [7] を用いた．実験では 1 つのレプリカから応答が返った時点でデータが公開される非同期レプリケーションを指定した．

3.2　実 験 結 果

端末間のネットワーク帯域が 1Gbps の場合の Producer のデータ送信スループットを図 4，Worker のデータ読み込みスループットを図 5，合計データ処理スループットを図 6 に示す．横軸がレプリケーション数を示し，縦軸が 1 秒あたりに処理できた画像の枚数を示す．図 4 から，非同期レプリケーションを用いているため Producer のデータ転送はレプリケーションによる影響はほぼなく，バッチサイズにより大きく変化した．バッチサイズが 1 の場合には転送スループットが大幅に低く，Worker のデータ読み込みのボトルネックになることがわかった．図 5 では，レプリケーション数の増加に伴うデータの読み込みスループットの低下が確認できる．また，バッチサイズを 5 以上にしても，性能の改善はあまり見られない．これは，データの転送とレプリカの配置が同時に行われているため，ネットワークの輻輳が起きていることが原因だと考えられる．図 6 から，合計データ処理においても，レプリケーション数が 1 の場合と 2,3 の場合でスループットの差が見られる．データ転送スループットの変化から Spark Streaming の 1 マイクロバッチ当たりのデータ量が減少していることが原因だと考えられる．

端末間のネットワーク帯域が 10Gbps の場合の Producer のデータ送信スループットを図 7，Worker のデータ読み込みスループットを図 8，合計データ処理スループットを図 9 に示す．図 4 と図 7 を比較すると，ネットワーク帯域を増やした場合にも Producer のスループットは向上していないことが確認でき

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 (4 コア) × 2 ソケット
Memory	8Gbyte

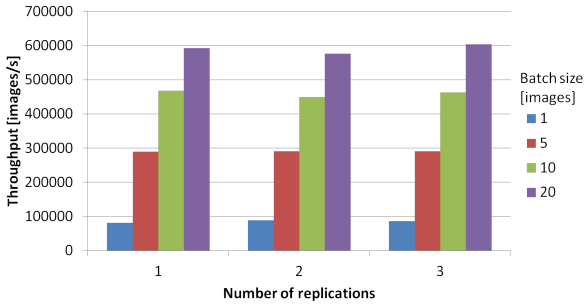

図 4　ネットワーク帯域が 1Gbps の場合の Producer のデータ送信スループット

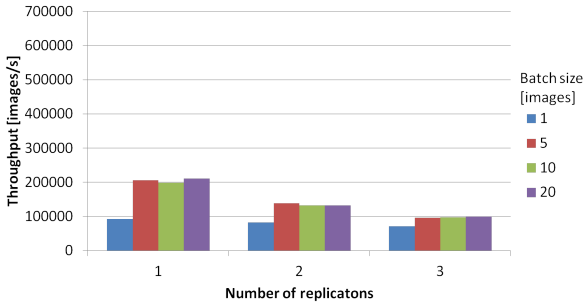

図 5　ネットワーク帯域が 1Gbps の場合の Worker のデータ読み込みスループット

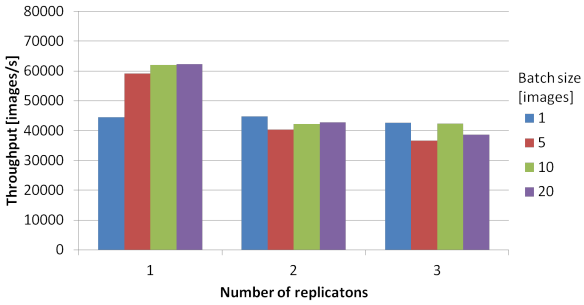

図 6　ネットワーク帯域が 1Gbps の場合の合計データ処理スループット

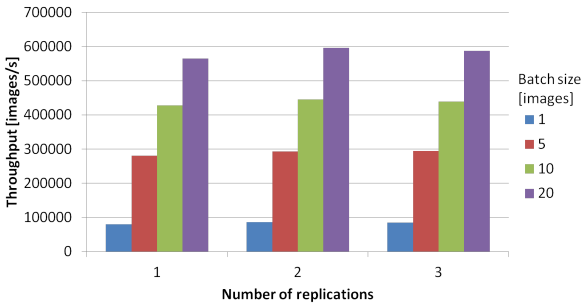

図 7　ネットワーク帯域が 10Gbps の場合の Producer のデータ送信スループット

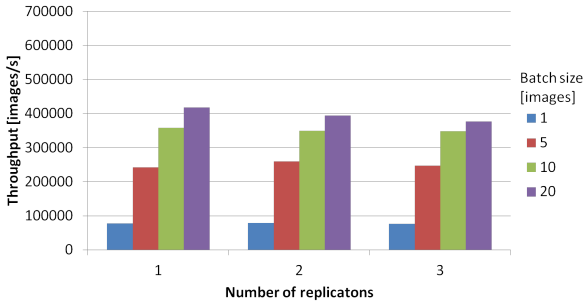

図 8　ネットワーク帯域が 10Gbps の場合の Worker のデータ読み込みスループット

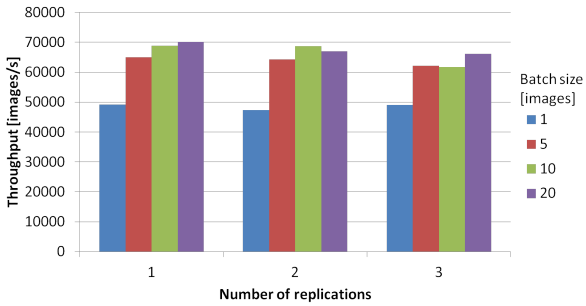

図 9　ネットワーク帯域が 10Gbps の場合の合計データ処理スループット

る．つまり，Producer と Broker 間は帯域が 1Gbps の場合でも輻輳が起きていないことがわかった．

一方，図 8 から，帯域を増やした場合に Consumer のスループットが向上していることが確認できる．Kafka のバッチサイズが 1 の場合には Producer がボトルネックとなっているため帯域による変化は見られないが，Kafka のバッチサイズが 5 以上に設定した場合はバッチサイズの増加に伴ってスループットも向上している．また，レプリケーション数の増加による性能劣化もほぼ見られない．図 9 から，Chainer を用いたデータ処理においてもレプリケーションによる性能劣化はほぼ見られなかった．バッチサイズが 5 以上の場合は，バッチサイズの変化によるスループットの向上はほぼ見られないため，Worker 側の処理がボトルネックとなっていることが原因だと考えらる．つまり，帯域を十分に増やすことにより，レプリケーション数の設定によるシステムの性能劣化を軽減し，Worker の処理能力に応じたスループットで解析処理を行うことができることが確認できた．

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 [11]．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 [12]．また，Kafka と Spark を用いたストリームデータ処理も近年数多く研究されている [13] [14]．Wang らは Kafka 上のメッセージキューと Spark Streaming ベースの処理エンジンを含む高性能なビッグデータ処理システムを設計し，Kafka と Spark Streaming 最適化システムのスケーラビリティについて議論している [15]．構造化および非構造化 Twitter データを抽出して分析するメモリ内処理機能を備えた分析フレームワークの開発を試みている [16]． Kafka がデータ取り込みタスクを実行し、Spark はデータ処理と機械学習アルゴリズムをリアルタイムで実行することを可能にする．我々は Kafka と Spark Streaming を用いて，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，Kafka と Spark Streaming を用いたリアルタイム動画像解析フレームワークを構築し，データの収集を行う Kafka のレプリケーションの設定による性能の変化を調査した．実験から，レプリケーション数を増やすことによりネットワー

クの輻輳が起き，データ転送のスループットが低下することが確認できた．データ転送のスループット低下により，システム全体のスループットが低下することがわかった．また，ネットワーク帯域を十分に増やすことにより性能劣化を軽減でき，レプリケーション数を増やした場合にも Worker の処理能力に応じたスループットで解析処理できることが確認できた．

今後の課題としては，他のパラメータの変化によるスループットへの影響の調査や，異なるデータセットを用いた場合の性能調査を検討している．

謝　　辞

この成果の一部は，JSPS 科研費 JP16K00177，平成２９年度国立情報学研究所公募型共同研究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] Ichinose, A., Takefusa, A., Nakada, H. and Oguchi, M.: A Study of a Video Analysis Framework Using Kafka and Spark Streaming, *Second Workshop on Real-time and Stream Analytics in Big Data* (2017).

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

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

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

[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).

[13] Stripelis, D., Ambite, J. L., Chiang, Y. Y., Eckel, S. P. and Habre, R.: A Scalable Data Integration and Analysis Architecture for Sensor Data of Pediatric Asthma, *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pp. 1407–1408 (2017).

[14] Park, J. and young Chi, S.: An implementation of a high throughput data ingestion system for machine logs in manufacturing industry, *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, pp. 117–120 (2016).

[15] Wang, M., Liu, J. and Zhou, W.: Design and Implementation of a High-Performance Stream-Oriented Big Data Processing System, *2016 8th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, Vol. 01, pp. 363–368 (2016).

[16] Yadranjiaghdam, B., Yasrobi, S. and Tabrizi, N.: Developing a Real-Time Data Analytics Framework for Twitter Streaming Data, *2017 IEEE International Congress on Big Data (BigData Congress)*, pp. 329–336 (2017).