「マルチメディア，分散，協調とモバイル
(DICOMO2018)シンポジウム」 平成30年7月

大規模データ分散処理基盤における パラメータ制御の一検討

加藤 香澄[1] 竹房あつ子[2] 中田 秀基[3] 小口 正人[1]

概要：近年カメラやセンサ等の発達やクラウドコンピューティングの普及により，一般家庭でのライフログの取得とそのデータの蓄積が可能になった．これらの技術は，遠隔地から家庭にいるお年寄りや子供，ペットを見守ることができる安全サービスや，防犯対策・セキュリティといった用途に応用されている．しかし，サーバやストレージを一般家庭に設置して取得・蓄積した動画像データの解析をするのは困難であるため，センサから取得した動画像データはクラウドに送信して解析する必要がある．ここで，動画像はデータサイズが大きいため，クラウドでの機械学習処理による動画像データ解析に要する計算量は大きくなる．また，クラウドには非常に多くの家庭からデータが送信されることが想定されるため，クラウドでの並列機械学習処理が必要不可欠である．
我々は，大規模データ分散処理プラットフォーム Apache Spark(以降，Spark と呼ぶ) を用いて，ディープラーニングフレームワーク Chainer による機械学習処理を並列化させ，そのパラメータを調整することで動画像データ解析処理の効率化を検討してきた．本稿では，Spark を用いた Chainer の並列処理と，分散処理フレームワーク Ray を用いた Chainer の並列処理について比較しつつ，処理の効率化について検討する．

A Study on Parameter Control in Large-scale Data Distributed Processing Infrastructure)

Kasumi Kato[1] Atsuko Takefusa[2] Hidemoto Nakada[3] Masato Oguchi[1]

1. はじめに

カメラやセンサ等の発達やクラウドコンピューティングの普及により，一般家庭でのライフログの取得とそのデータの蓄積が可能になった．この技術は遠隔地から家庭にいるお年寄りや子供，ペットを見守ることができる安全サービスや，防犯対策・セキュリティといった用途に応用されている．しかし，サーバやストレージを一般家庭に設置して取得・蓄積した動画像データの解析をするのは困難なため，センサから取得した動画像データをクラウドに送信して解析する必要がある．動画像はデータサイズが大きいためセンサ・クラウド間の通信量が膨大になり，クラウドでの機械学習処理に要する計算量も膨大になる．また，クラウドには非常に多くの家庭からデータが送信されることが想定されるため，クラウドでの処理時間が長くなってしまう．従って，クラウドでの動画像データ処理を並列化して高速化を図る必要がある．

我々は，大規模データ分散処理プラットフォーム Apache Spark[1] を用いて，ディープラーニングフレームワーク Chainer[2] による機械学習処理を並列化させ，そのパラメータを調整することで動画像データ解析処理の効率化を検討してきた [15]．Spark には様々な機能が備わっており，それらが密接に関わっているため，効率の良く分散処理を行わせるのが困難であった．また，昨年新たに Ray[14] という分散処理基盤が公開された．Ray は大規模機械学習に向けて開発されており，まだ機能が複雑化していないため，Ray を用いての処理の効率化を視野に入れることにした．

本稿では，Ray を用いた Chainer による機械学習処理の並列化について検討するとともに，Spark による Chainer の機械学習処理の並列化と比較する．Spark と Ray による分散処理の挙動を調査する実験をそれぞれ行い，その結果

1 お茶水女子大学
2 国立情報学研究所
3 産業技術総合研究所

から処理を効率化・高速化する手法について考察する．

2. 関連技術

本研究ではクラスタでの負荷分散の基盤として Spark，Ray，動画像データの解析処理に Chainer を用いる．以下に各ソフトウェアの概要を述べる．

2.1 Apache Spark

Spark は，高速かつ汎用的であることを目的に設計されたクラスタコンピューティングプラットフォームである．カリフォルニア大学バークレー校で開発が開始され，2014 年に Apache Software Foundation に寄贈された．マイクロバッチ処理という極小単位でのバッチ処理を行うことが可能である．演算をオンメモリで行うため，アプリケーションがメモリ内にデータを保存し，高コストなディスクアクセスを避けて処理全体の実行速度を向上させることができる．

Spark プロジェクトは，Spark コアと構造化データを扱う Spark SQL，ライブストリーム処理を実現する Spark Streaming，一般的な機械学習の機能を含むライブラリである MLlib，グラフ処理を担う GraphX といった複数のコンポーネントで構成されている [3]．Spark コアにはタスクスケジューリング，メモリ管理，障害回復，ストレージシステムとのやりとりといった Spark の基本的機能が備わっている．利用者が，Spark コアで定義されている RDD(Resilient Distributed Dataset) にデータを保持し，用意されているメソッドを用いて処理を記述することで自動的に分散処理される．RDD とメソッドの組み合わせによって繰り返しの機械学習，ストリーミング，複雑なクエリ，そしてバッチなど幅広い領域を簡単に表現できる．Hadoop[4] などの他のビックデータのツールとの組み合わせが可能であり，優れた汎用性を備えている．

2.2 Ray

Ray は，大規模機械学習や強化学習アプリケーションを対象とした分散実行フレームワークである．Spark の開発者たちによって 2017 年に開発された．Ray は Python と C++で開発された分散処理エンジンである．1 つのノードにつき 1 つのオブジェクトストアが共有メモリに不変オブジェクトを格納し，最小限のコピーとデシリアライゼーションで同じノード上のオブジェクトを効率的に共有できる．ボトムアップ分散スケジューラを用いており，1 つのグローバルスケジューラと各ノードに 1 つのローカルスケジューラでスケジューリングを行う．

Ray の分散実行フレームワーク上には Ray RLib という強化学習実行ツールキットが構築される．RLlib は PyTorch，TensorFlow，Teano などのフレームワークで書かれた既存の強化学習アルゴリズムを用いており，様々なトレーニング戦略を簡単に使用できる．

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，「Flexible(柔軟性)」「Intuitive(直感的)」「Powerful(高機能)」の 3 つの特徴を掲げている．シンプルな記法で直感的にコードを記述できる点や，CUDA をサポートしており GPU による高速演算が可能である点，インストールが簡単である点も大きな特徴である．画像処理，自然言語処理，ロボット制御など幅広い分野に用いられている．

多くのディープラーニングフレームワークは一度ニューラルネット全体の構造をメモリ上に展開し，その処理を順に実行して順伝播・逆伝播を行うアプローチを取っている．一方，Chainer は実際に Python のコードを用いて入力配列に何の処理が適用されたかだけを記憶しておき，それを誤差逆伝播の実行に利用する．このアプローチにより，畳み込みやリカレントなどの様々なニューラルネットや複雑化していくディープラーニングにも対応している．

3. 実験

本研究では図 1 のような大規模データ分散処理基盤を想定している．各家庭に設置されたセンサやカメラから動画像データがクラウドに送信されると，ストリーミング処理基盤によって動画像データが処理され，分散処理基盤に渡される．分散処理基盤がデータを受け取るとディープラーニングフレームワークによるデータの解析処理が行われ，結果がユーザに返される．図 2 にクラウド内のフレームワークを示している．ストリーミング処理基盤としては Apache Kafka[16] の採用を検討しており，Kafka クラスタから Spark もしくは Ray クラスタにデータが渡され，クラスタ内のワーカにおいて Chainer の機械学習処理が行われる．本稿では，Kafka によるストリーミング処理は考慮せず，Spark と Ray を用いた Chainer の認識処理部分に着目して計測を行い，その性能を比較する．

3.1 実験概要

実験では，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[5] を用いる．

図 3 に Spark を用いた場合のマスタ・ワーカ処理を示している．マスタで Python のプログラムを実行し，MNIST を Spark に読み込ませて RDD を作成する．作成した RDD をワーカに渡して，ワーカにて Chainer を用いた MNIST の評価を行う．各端末は Spark Standalone Mode で接続した．

図 4 に Ray を用いた場合のマスタ・ワーカ処理を示して

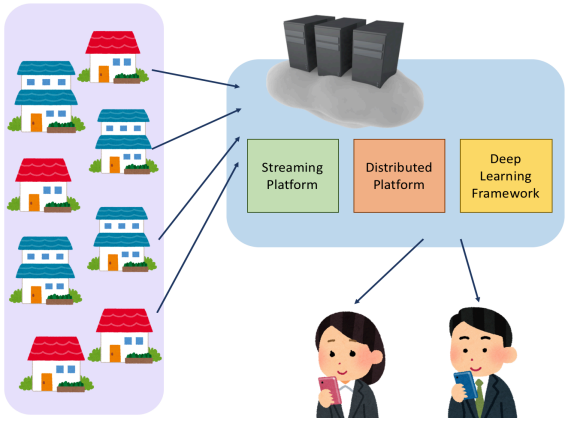

図 1　想定する大規模データ分散処理基盤

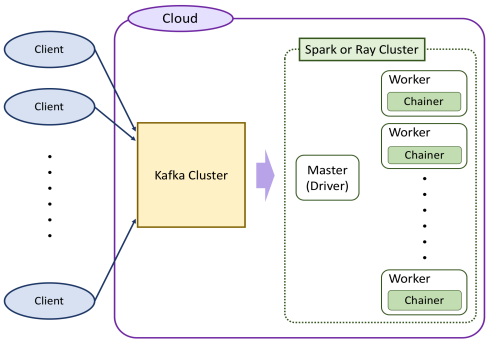

図 2　クラウド内の想定フレームワーク

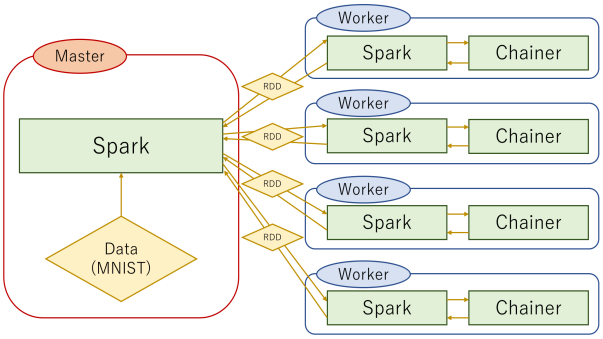

図 3　Spark と Chainer を用いたマスタ・ワーカ処理

いる．マスタで Python のプログラムを実行するとマスタノード内の Ray ドライバが自身のノードのオブジェクトストアにデータを置き，ローカルスケジューラに連絡する．ローカルスケジューラは処理が自身のノード内で完了できると判断するとノード内でスケジューリングを行うが，他のノードにも処理を分散させたいと判断するとグローバルスケジューラに伝達する．グローバルスケジューラは各ノードのローカルスケジューラに命令を送り，ワーカが共有オブジェクトストアのデータをもとに Chainer を用いて MNIST の評価を行う．

Spark と Ray それぞれの場合について，プログラムを実行してから各ワーカで評価が行われ，マスタに結果が返ってくるまでに要する時間を計測した．

実験で用いた計算機の性能を表 1 に示す．マスタ及び 1

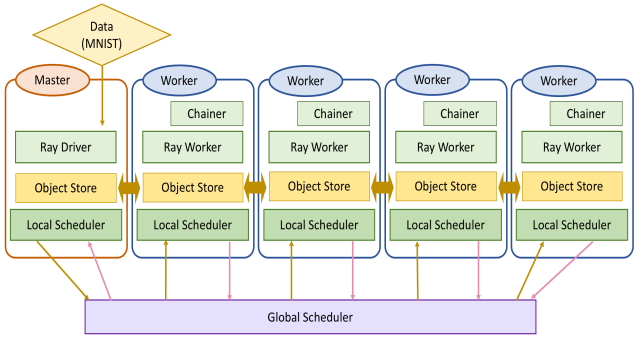

図 4　Ray と Chainer を用いたマスタ・ワーカ処理

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	48Gbyte

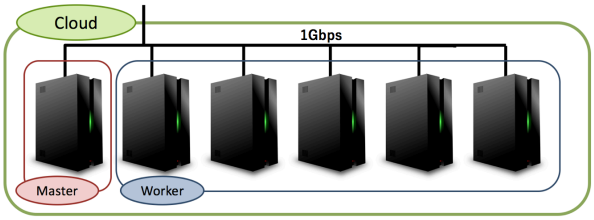

図 5　実験環境

〜5 台の全ワーカには同質のノードを用いており，図 5 に示すように 1Gbps のネットワークで接続されたクラスタ構成とした．

3.2　実験結果

1000 個タスクを用意しノード数を 1〜5，パーティション数を 8〜96 まで 8 刻みで変化させた際の実行時間の計測結果 10 回の平均値を調査した．パーティション数とは，1000 個のタスクの塊をいくつに分割してワーカに分配するかを決定するパラメータであり，一般的には値が大きいほどワーカ間の偏りが少なく分配できるが，その分通信のボトルネックが大きくなる．

図 6 に Spark を用いた際の実行時間を示す．図から，ノード数の増加に応じて実行時間が減少していることが伺える．ノード数 1 の時とノード数 5 の時を比較すると，処理時間は 1/3 ほどまで短縮されたことがわかった．しかし，パーティション数に応じた実行時間の変化は少なく，パーティション数 40 ほどで結果が横ばいになった．

Spark と同様に，1000 個タスクを用意しノード数を 1〜5 と変化させた際の Ray の実行時間の計測結果 10 回の平均値を図 7 に示す．Ray では Spark のようなパーティションの設定がないため，データはワーカ 1〜5 に対してラウンドロビンに分配している．図から，ノード数が増加するごとに実行時間が減少している．ノード数 1 の時とノード

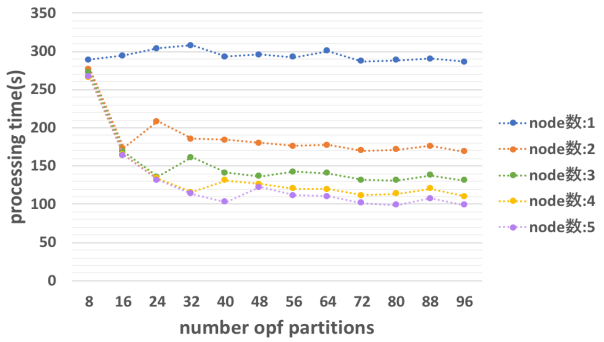

図 6　Spark を用いた際の実行時間の変化

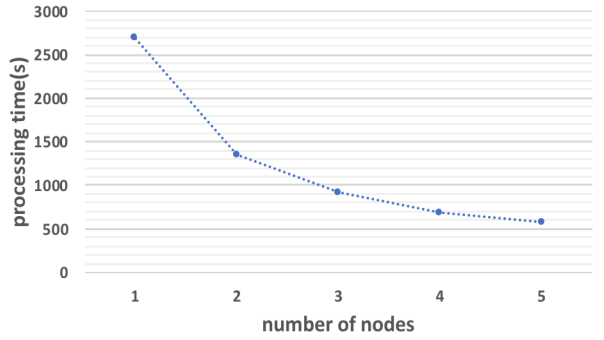

図 7　Ray を用いた際の実行時間の変化

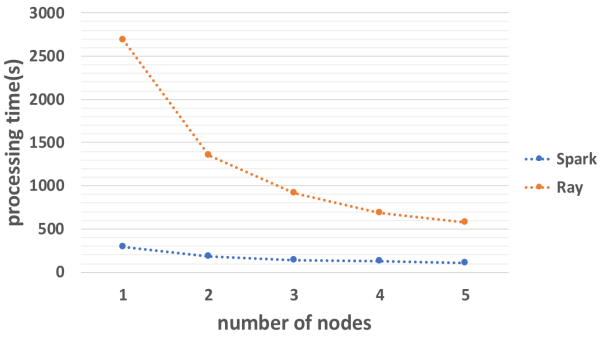

図 8　Spark，Ray それぞれにおける実行時間の変化

表 2　Spark，Ray それぞれおけるタスクごとの処理時間

実験	先頭タスク	先頭以降のタスク
Spark	約 6s	約 2s
Ray	約 5.5s	約 2.2s

数 5 の時を比較すると，処理時間は 1/5 ほどまで短縮されたことがわかった．

図 8 に Spark でのパーティション数 40 での計測結果と Ray での計測結果を示す．両結果を比較すると，Spark の方が処理が高速であることがわかる．また，それぞれの実験においてタスク 1 つあたりの処理にかかった時間を表 2 に示す．表から，タスクごとの処理時間には差がないが，Ray における全体の処理時間が極端に遅くなっていることがわかった．

4. 関連研究

柳瀬ら [7] の研究では，MapReduce[8][9] モデルに Map と Reduce の反復やデータ型の制限を加えることで，機械学習の並列化に最適化した分散計算プラットフォームが提案されている．評価実験において提案手法の高速性とスケーラビリティが示されているが，膨大なデータ量を扱う際の挙動が課題となっている．

伍ら [10] の研究では，非決定情報システム (NIS) でラフ集合理論の構築を通して，不完全なデータも対象とするデータマイニング手法の研究 [11] における，Spark のクラスタコンピューティング機能を用いた処理の高速化がなされている．

分散処理により Chainer を用いた学習処理を高速化する ChainerMN[13] が Chainer の追加パッケージとして開発された．既存の Chainer の学習コードから数行変更することで通常の学習に All-Reduce のステップを追加し，全ワーカが求めた勾配を利用することで学習の高速化がなされている．

本研究では，一般的なデータ処理インフラストラクチャを用いた機械学習の並列化処理の高速化を目指している．

5. まとめと今後の予定

Chainer による認識処理を分散処理基盤である Spark と Ray の 2 種類それぞれで並列化し，その性能の調査を行った．結果から，現段階では Spark が Ray より高速に処理を行えることがわかった．

今後は，Ray の処理の高速化を図るとともに，将来的に Kafka と組み合わせたストリーミング処理を考慮した実験を行っていく．

参考文献

[1] Apache Spark, https://spark.apache.org/.
[2] Tokui, S., Oono, K., Hido, S. and Clayton, J.: Chainer: a Next-Generation Open Source Frame- work for Deep Learning, In Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Informa- tion Processing Systems (NIPS) (2015). 6 pages.
[3] Karau, H., Konwinski, A., Wendell, P. and Zaharia, M.: Learning Spark, Cambridge: O'Reilly Media (2015).
[4] Apache Hadoop, https://hadoop.apache.org/.
[5] Lecun, Y., Cortes, C. and Burges, C. J.: The MNIST Database of handwritten digits, http://yann.lecun.com/exdb/mnist/.
[6] Chiu, D.-M. and Jain, R.: Analysis of the increase and decrease algorithms for congestion avoidance in computer networks, Computer Networks and ISDN Systems, vol. 17, pp. 1-14 (1989).
[7] Yanase, T., Hiroki, K., Itoh, A. and Yanai, K.: MapReduse Platform for Parallel Machine Learing on Large-scale Dataset, Transactions of the Japanese Society for

Artificial Intelligence, vol. 26, No. 5, pp. 621-637 (2011).
[8] Dean, J. and Ghemawat, S.: MapReduse: Simplified Data Processing on Large Clusters, OSDI'4: Sixth Symposium on Operating System Design and Implementation (2004).
[9] Dean, J. and Ghemawat, S.: MapReduse: simplified Data processing on large clusters, Communications of the ACM, vol. 51, No. 1, pp. 107-113 (2008).
[10] Wu , m., Yamaguchi, N., Nakata, M. and Sakai, H.: Improving the Performance of NIS-Apriori Algorithm by Parallel Processing, Proceedings of the Fuzzy System Symposium, vol. 30, pp. 592-595 (2014).
[11] Sakai, H. Okuma, H., Wu M., Nakata, M.: Rough nondeterministic information analysis for uncertain information, The Handbook on Reasoning-Based Intelligent Systems, World Scienti c, pp. 81-118 (2013).
[12] Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S., and Stoica, I.: Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In EuroSys 2010, (2010).
[13] ChainerMN, https://chainermn.readthedocs.io/en/latest/index.html.
[14] Ray，http://ray.readthedocs.io/en/latest/index.html .
[15] 加藤香澄，竹房あつ子，中田秀基，小口正人: ビッグデータ分散処理基盤におけるパラメータ制御の一検討，DEIM2018，G1-5 (2018) .
[16] Apache Kafka，https://kafka.apache.org .