大規模データ分散処理プラットフォーム Apache Sparkを用いた分散並列機械学習に関する考察

加藤　香澄† 　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: †{g1320510,oguchi}@is.ocha.ac.jp, ††takefusa@nii.ac.jp, †††hide-nakada@aist.go.jp

あらまし　近年，お年寄りや子供を見守るサービスや防犯カメラなどによるライフログの利用が普及し，多様に活用されるようになってきているが，動画像解析に要する通信量や計算量，プライバシーに関する問題が介在している．また，近年ディープラーニングの技術が非常に発達してきており，画像認識や音声認識を始めとする様々な分野に応用されている．しかし正確な認識処理を行うためには大量のデータ処理が必要となるため，処理の並列化が求められる．そこで本研究では，ディープラーニングフレームワークである Chainer を，クラスタコンピューティングプラットフォームである Apache Spark 上で動作させることによる，分散並列機械学習処理に関して考察する．

キーワード　分散処理，並列処理，機械学習，Spark，Chainer

Consideration on Distributed Parallel Machine Learning using Apache Spark,a Large-scale Data Distributed Processing Platform

Kasumi KATO†, Atsuko TAKEFUSA††, Hidemoto NAKADA†††, and Masato OGUCHI†

† Ochanomizu University　2-1-1 Otsuka,Bunkyo-Ku, Tokyo 112–8610, Japan
†† National Institute of Informatics　2-1-1 Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430, Japan
††† National Institute of Advanced Industrial Science and Technology (AIST)　1-1-1 Umezono, Tsukuba, Ibaraki 305–8560, Japan
E-mail: †{g1320510,oguchi}@is.ocha.ac.jp, ††takefusa@nii.ac.jp, †††hide-nakada@aist.go.jp

1. はじめに

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ので，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センサ・クラウド間の通信量が膨大になってしまう．また，クラウドにて機械学習処理による動画像データ解析を行うとその解析に要する計算量も膨大になるので，クラウドでの負荷が非常に大きくなる．従ってこの問題については，動画像データの機械学習処理を並列化してクラウドの負荷を軽減する必要がある．

本研究では，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) [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ることで，動画像データ解析処理の効率化を図る．

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，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特徴であり，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でき，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 上では 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操作することで自動的に分散が可能である．この RDD の本拠地が Spark コアで，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．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が密接に結合して構成されている．

Spark は前述の 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．これにより Spark ジョブは，Hadoop ジョブに比べて 1/2 から 1/10 のコード量で最大で 100 倍もの速度で実行することが可能である．

2.2 Chainer

Chainer は Preferred Networks が開発したディープラーニングの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，シンプルな記法で直感的にコードを記述できる．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，その処理を順に見てその通りに順伝播・逆伝播を実行するというアプローチを取っているのに対して 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．CUDA をサポートしており，GPU による高速演算が可能であるのも大きな特徴である．

3. 実　　験

本稿では，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 [4] を用いて実験を行う．マスタで Python のプログラムを実行することにより，MNIST を Spark に読み込ませて RDD に変換し，同プログラム上で Chainer を呼び出して RDD を渡し，ワーカにて評価を行う実験 (図 1) を Spark の分散機能を利用して実施した．

3.1 実験概要

実験では，マスタ 1 台とワーカとして最大 5 台の端末を Spark Standalone Mode で接続し，マスタでプログラムが実行され，各ワーカでのタスクが完了してワーカからマスタに結果が返って出力されるまでに要する時間を測定した．本実験では，以下 2 つのパラメータを変えて測定した．

（1） Spark に読み込ませるデータの partition 数
（2） ワーカのノード数

また，この実験を元にタスクがどのように各ノードに分配されているのかを観測した．

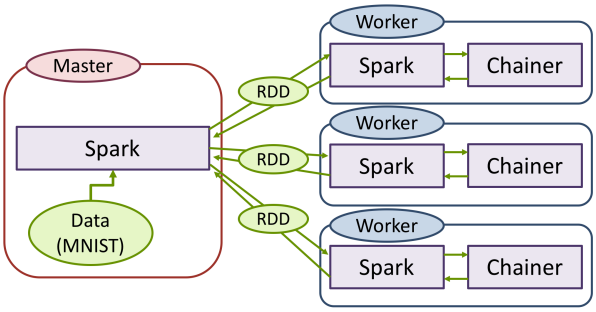

図 1　Spark と Chainer を用いたマスタ・ワーカ処理

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	8Gbyte

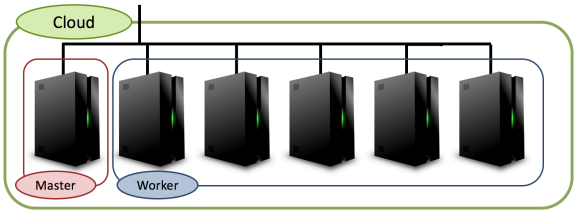

図 2　実 験 環 境

実験で用いた計算機の性能を表 1 に示す．マスタ及び全ワーカには同質のノードを用いており，図 2 に示すクラスタ構成とした．

3.2 実験結果

測定結果を図 3 に示す．このグラフは，ノード数を 1 〜 5 まで変化させ，partition 数を 2,3,5,10,20,30,40,50 と変化させた際の測定結果 10 回の平均値を用いて作成している．実験の結果，partition 数が増加すると実行時間が約 1/2 ほどまで減少することがわかった．また，partition 数の増加による実行時間の減少は partition 数 40 ほどで横ばいになった．一方，ノード数の増加による実行時間の減少はわずかであり，効率よく分散処理が行えていないことがわかった．

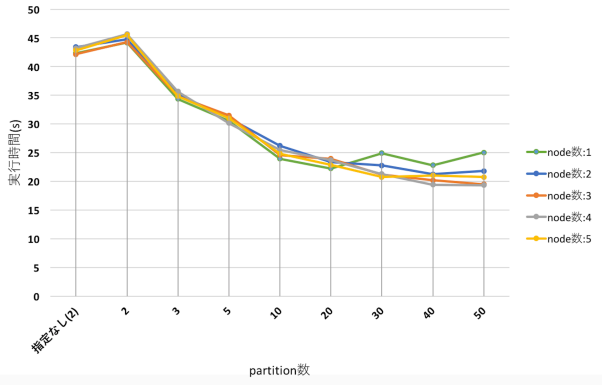

図 3　partition 数及びノード数の変化による実行時間

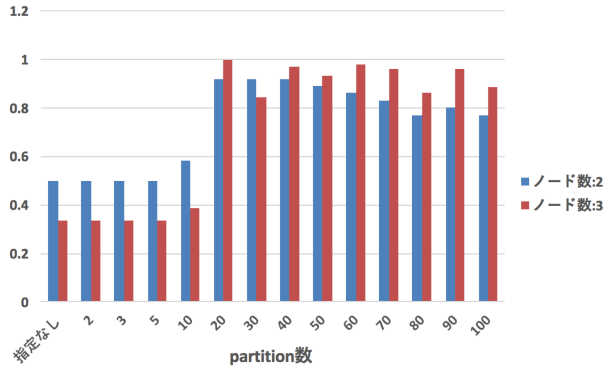

図 4　Fairness Index

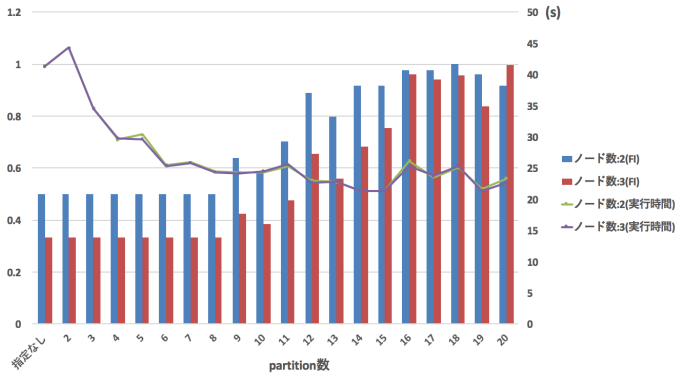

図 5　Fairness Index と実行時間変化

図 4 に，ノード数が 2 と 3 の場合についてタスクがどのように各ノードに分配されているのかを観測した結果を，公平性を示す指標である Fairness Index [5] で示す．Fairness Index は以下の式で計算でき，値が 1 に近いほど公平性が高いことを示す．

$$FairnessIndex: f_i = \frac{(\sum_{i=i}^{k} x_i)^2}{\sum_{i=i}^{k} x_i^2}$$

結果から，実行時間に減少が見られた場合でも実際にはタスクが偏って分配されてしまっていたことが判明した．グラフより，partition 数 10 から 20 の間で公平性に大きな変化が見られたため，partition 数 20 までの Fairness Index 及び実行時間変化を図 5，ノードに割り当てられた最多タスク数及び実行時間変化を図 6 に示す．図 5 と図 6 から，公平性と実行時間は必ずしも反比例にならず，ノードに割り当てられた最多タスク数はノード数が 2 と 3 の場合でほぼ同じになっていた．

さらに上記の結果を踏まえ，ノードごとにどのようなタイミングでジョブが与えられているのか調査した．特に，partition 数指定なし，partition 数 2〜3，と partition 数 16〜20 の 2 つの区間に着目した．

上記の 2 つの区間について，ノード数を 3 として計測した結果を図 7 と図 8 に示す．partition 数の指定がない場合にはデフォルトで partiton 数が 2 になるが，ここで partition 数の指定がない場合と partition 数 2 の場合で最初のジョブ割り当て時間に差が生じるのは，メソッドを用いて partition を設定するのに時間を要するためだと考えられる．partition 数 3 の場合，分散処理による実行時間の減少の影響がメソッドの利用に

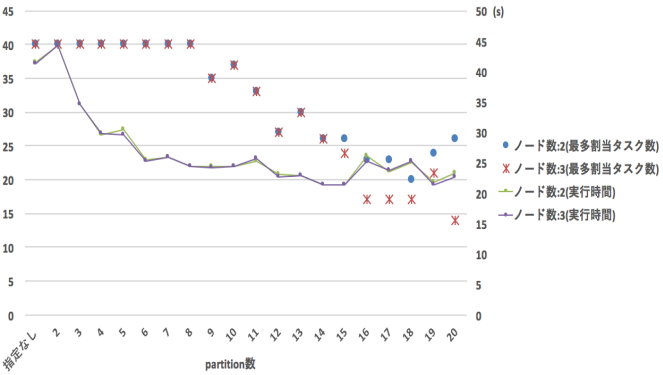

図 6　最多割当タスク数と実行時間変化

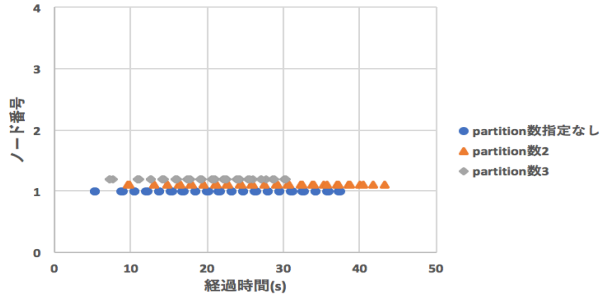

図 7　ジョブ割当タイミング (partition 数指定なし，2，3)

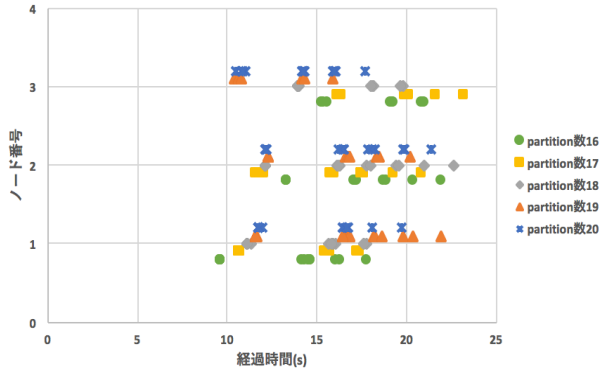

図 8　ジョブ割当タイミング (partition 数 16〜20)

よる実行時間増加の影響を上回り，処理全体として実行時間の減少に成功している．(図 7)

partition 数 16〜20 になるとジョブ割り当てのタイミングに大きな違いは現れなかったが，最初のジョブ割り当ての時間に注目すると partition 数が多いほど割り当てまでにわずかに時間を要することが伺え，その時間は 10 秒前後であった．(図 8)

以上のことから，ある程度の実行時間減少・タスク分配の公平性をもって本研究における分散並列処理を行うには RDD 作成・partition 設定などに関して 10 秒前後の時間を要することがわかった．また，効率的な分散処理を行えた際の実行時間の目安として

$$\frac{\text{42 秒 (初期状態の実行時間)} - \text{10 秒 (}RDD\text{ 作成時間等)}}{\text{ノード数}}$$

が推測できる．

4. まとめと今後の予定

Chainer による解析処理を Spark で並列化し，負荷分散を行った．実行時間とタスクとジョブの割り当てに関して実験し，ノード数増加による実行時間の減少はわずかであること，タスク割り当ての公平性と実行時間が反比例しないことに加え，ある程度実行時間を減少させてノード間の公平性を保ちつつ処理を行うためにはワーカでの評価を開始する前に 10 秒ほどの処理時間を要することがわかり，本研究における並列処理の目安となる実行時間が推測できた．

今後の課題として，ふるまいの実態をより詳しく調査していくことにより現時点での並列処理の課題を明らかにし，ノード数増加による実行時間の減少を目標とした処理の効率化を図る．

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] 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] Apache Hadoop, https://hadoop.apache.org/.
[4] Lecun, Y., Cortes, C. and Burges, C. J.: The MNIST Database of handwritten digits, http://yann.lecun.com/exdb/mnist/.
[5] 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).