6U-09

ビックデータ分散処理基盤を用いたタスク並列化におけるパラメータ制御の考察

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 ‡ 国立情報学研究所 § 産業技術総合研究所

1. はじめに

カメラやセンサ等の発達により，一般家庭でのライフログの取得が可能になり，活用されるようになった．サーバやストレージを一般家庭に設置して解析するのは困難であるため，取得した動画像データはクラウドで解析する必要がある．しかし，動画像はデータサイズが大きく，その解析に要する計算量も膨大になるので，クラウドにおける負荷も大きくなる．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検討してきた [4]．本稿では特に，Spark のパラメータ制御を行うことによるタスク並列化について考察する．

2. 関連技術

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．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，他のビックデータのツールと密接に組み合わせることができる．Spark 上では Resilient Distributed Dataset(RDD) 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．

2.2 Chainer

Chainer はニューラルネットワークを実装するためのライブラリで，シンプルな記法によりネットワークを直観的に記述でき，畳み込みやリカレントなどの様々なニューラルネットにも対応可能なことが特徴である．このニューラルネットを多層にしたものはディープラーニングと呼ばれ，画像認識・自然言語処理・音声認識など様々な分野に応用されている．

3. 実験

本稿では，マスタで Python のプログラムを実行して，MNIST を Spark に読み込ませて RDD に変換し，同プログラム上でワーカにおいて Chainer を呼び出して RDD を渡し，その評価を行う実験 (図 2) を Spark の分散機能を利用して実施した．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[3] を用いた．

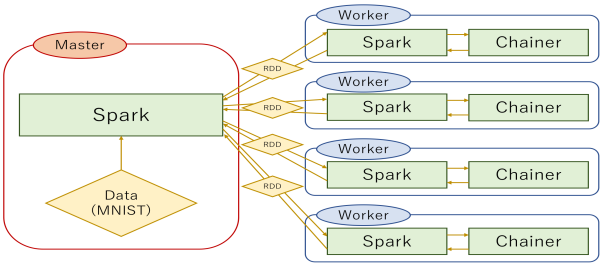

図 1: Spark と Chainer を用いたマスタ・ワーカ処理

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 (4 コア) × 2 ソケット
Memory	8Gbyte

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，Spark に読み込ませるデータのパーティションの作成に関するパラメータを以下 3 つの手法で設定し，その性能の調査を行った．

1. メソッド partitionBy() を利用
2. メソッド repartition() を利用
3. partitionBy() においてパーティショナを調整

partitionBy() と repartition() はどちらとも Spark に備わっているメソッドであり，partitionBy() はパーティション数とパーティショナを，repartition() はパーティション数を引数にとる．手法 3 では partitionBy() の引数としてタスクをラウンドロビン的に分配できるようなパーティショナを与える．

また，各手法において Spark に読み込ませるデータのパーティション数とワーカのノード数をそれぞれ変化させて測定した．実験で用いた計算機の性能を表 1 に示す．マスタ及び全ワーカには同質のノードを用いており，図 2 に

Study on Distributed Parallel Machine Learning using Apache Spark, a Large-scale Data Distributed Processing Platform
Kasumi Kato†
Atsuko Takefusa‡
Nakada Hidemoto§
Masato Oguchi†
†Ochanomizu University
‡National Institute of Informatics
§National Institute of Advanced Industrial Science and Technology (AIST)

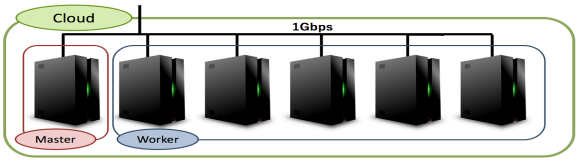

図 2: 実験環境

図 3: partitionBy() を用いた際の実行時間の変化

示す 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 個タスクを用意し各設定でノード数を 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測定結果を 3 回の平均値で示す．

3.2.1 メソッド partitionBy() の利用

測定結果を図 3 に示す．実験の結果，ノード数の増加により実行時間が 1/3 ほどまで減少することがわかった．また，パーティション数 32 ほどで実行時間が横ばいになった．

3.2.2 メソッド repartition() の利用

測定結果を図 4 に示す．ノード数増加による実行時間の減少は見られたが，指定パーティション数 8,16,32,64 で実行時間が極端に遅くなっていた．これは，パーティション内のタスク数が 0 であるパーティションが多数できてしまったことによるもので，例として指定パーティション数 32 のときパーティション内のタスク数は 560 が 1 個，440 が 1 個，0 が 30 個になっていた．

3.2.3 パーティショナの調整

測定結果を図 5 に示す．図から，ノード数 3 以上で実行時間が一致し，結果が横ばいになることがわかった．また，この調整によりパーティション数の指定に関わらず，パーティション内のタスク数は 18 が 55 個，10 が 1 個とほぼ均等になっていたが，ノード数 4 以上の場合でもノードは 3 つしか使われなくなってしまうため，図のような結果となっていた．

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．データのパーティションの作成に関する設定を行い，partitionBy() と repartition() という 2 つのメソッド

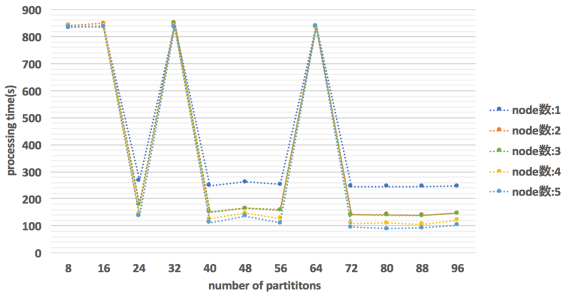

図 4: repartition() を用いた際の実行時間の変化

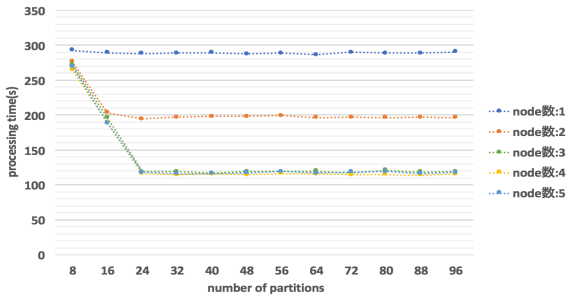

図 5: パーティショナを調整した際の実行時間の変化

の挙動の違いを確認するとともに，partitionBy() におけるパーティショナの調整を試みた．

今後の課題として，調整したパーティショナで切り分けたパーティションを効率よく複数ノードに振り分ける手法を検討する．

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．

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 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.

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

[4] 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を用いた機械学習処理並列化の一考察，マルチメディア，分散，協調とモバイル DICOMO2017 シンポジウム，p. 796-802 (2017).