ディープラーニングを用いた リアルタイムライフログ解析フレームワークにおける負荷分散の実装と評価

理学専攻・情報科学コース　一瀬 絢衣 (指導教員：小口正人)

1　はじめに

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．

本研究では，Kafka と Spark Streaming を用いた，複数カメラからの動画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する．本稿では，スループットの計測から提案フレームワークの評価を行った．実験から，解析処理を行う端末数や，データの保存を行う Broker のノード数によって全体のスループットが変化することが確認できた．また，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，データ転送やシステム全体のスループットが低下す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について述べる．

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 [1]．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 [2]．Producer からデータを収集し，Consumer にデータを公開する．データを保持する Broker は，複数のサーバ上でクラスタとして実行可能であり，トピックと呼ばれるカテゴリにデータを格納する．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．非同期レプリケーション，同期レプリケーションの設定が可能であり，デフォルトでは 1 つのレプリカから応答が返った時点で Consumer にデータが公開される同期レプリケーションに設定されている．

2.3　ストリーミング機械学習フレームワーク

本研究では図 1 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 でデータが保持され，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，データをディープラーニングのフレームワークである Chainer [3] の要求する型へ変換し，Chainer を用いて識別処理を行う．

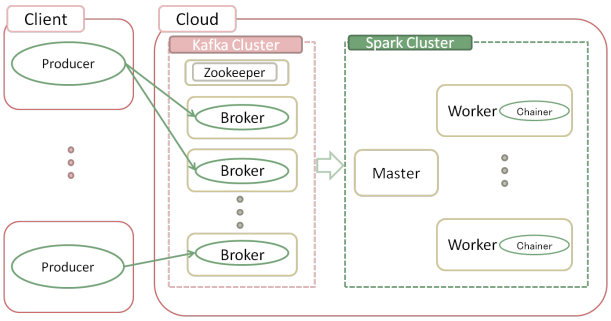

図 1: ストリーミング機械学習

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 (4 コア) × 2 ソケット
Memory	8Gbyte

3　スループット計測

図 1 に示すクラスタ構成で，スループットを計測する．黄色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，その間のネットワーク帯域は 1Gbps となっている (表 1)．Kafka クラスタ，Spark クラスタ共に最大 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 台である．Producer は Broker と同じノード内で作動させ，合計 16 個の Producer を動かす．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 を用いた．

3.1　Broker 数，Worker 数の変化によるスループットの変化

3.1.1　実験概要

データの保存を行う Kafka の Broker，データを処理する Spark の Worker の数をそれぞれ 1，2，4 と変化させた．データのパーティション数は 16，32 とした．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．ここでは，Producer は画像データを 1 枚ずつ送信し，Worker 内でデータを 100 枚にまとめ，100 枚ごとに Chainer を呼び出す．Producer からのデータ転送と同時に Worker でデータを読み込み

図 2: Partition 数 16 の場合の Broker 数，Worke 数の変化によるスループットの変化

図 3: Partition 数 32 の場合の Broker 数，Worke 数の変化によるスループットの変化

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結果に用いた．

3.1.2 実験結果

Partition 数を 16 に設定した場合の結果を図 2，32 に設定した場合の結果を図 3 に示す．横軸が Broker 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．Partition 数が 16 の場合には，Worker 数が 4 の場合に，2 の場合と比べてスループットの向上があまり見られない．これは，使用している端末の CPU が 4 コア× 2 ソケットであるため，Worker 数が 2 のときに Worker のコア数の合計が Partition 数と一致しているためである．Parttion 数以上に Worker 数を増やしても Worker 全体で使用されるコア数が変化しないためである．このとき，Worker 側の処理がボトルネックとなっているため，Broker 数の変化によるスループットの変化はほぼ見られない．
一方 Partition 数が 32 の場合には Worker 数が 4 の場合にコア数の合計が Partition 数と一致するため，Worker 数が 4 の場合と比較してスループットが向上している．Worker 数が 4 の場合に，Broker 数が 2 のときと比べて 4 のときの方がスループットが高いことから，Worker 数 4，Broker 数 2 の場合には Broker 側が全体のボトルネックとなっていることが確認できた．

3.2 レプリケーションによるスループットの変化

3.2.1 実験概要

Broker 数，Worker 数はそれぞれ 4 台に固定した．それぞれの Producer で 10 万枚のデータを送信し，合計 160 万枚のデータを処理する．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．Worker 内でまとめるデータは 100 枚に固定し，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，節 3.1 と同様，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結果に用いた．

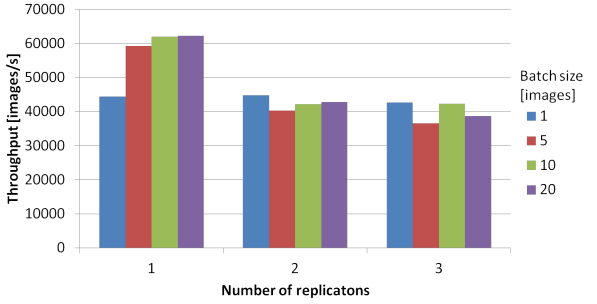

図 4: レプリケーション数の変化によるスループットの変化

3.2.2 実験結果

結果を図 4 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．結果から，レプリケーション数が 1 の場合には Kafka のバッチサイズの増加に伴いスループットも向上していることが確認できた．しかし，レプリケーション数が 2,3 の場合にはスループットの向上は見られない．レプリケーションによるネットワークの輻輳からデータ転送のスループットが低下し，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．

4 まとめと今後の課題

本研究では，Kafka と Spark Streaming を用いた動画像解析フレームワークを構築し，用いるノード数やパラメータの変化による性能の変化を調査した．実験から，実験から，解析処理を行う端末数や，データの保存を行う Broker のノード数によって全体のスループットが変化することが確認できた．また，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，データ転送やシステム全体のスループットが低下す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．

参考文献

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

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

[3] 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.