複数分析処理におけるMapReduce最適化

福本　佳史† 　鬼塚　真†

† 日本電信電話株式会社 NTT サイバースペース研究所　〒 239–0847 神奈川県横須賀市光の丘 1-1
E-mail: †{fukumoto.yoshifumi,onizuka.makoto}@lab.ntt.co.jp

あらまし　本研究では，大規模なデータセットの分析を分散処理によって高速に行う問題を扱う．既存技術では MapReduce と機械学習アルゴリズムの組み合わせにより大規模データに対する分析処理を実現可能だが，最適な学習結果を得るためのハイパーパラメータ値調整に伴い，長時間の処理を何度も繰り返すという問題がある．そこでパラメータ値のみが異なる複数の MapReduce 処理における共有化手法を提案する．本手法は，（1）パラメータを利用するクラス・関数の検知，（2）検知した情報を用いた複数 MapReduce 処理の重複範囲の特定・共有化による処理量削減，により構成される．本手法を機械学習ライブラリ Mahout の SVM アルゴリズムを用いた MapReduce 処理に適用したところ，重複範囲の割合に応じて最大で 2 倍の高速化を確認した．また NavieBayes アルゴリズムを用いた実験により，MapReduce ジョブ単位での共有化による高速化を机上で確認した．

キーワード　クラウドコンピューティング, MapReduce, マルチクエリ最適化, 機械学習

Optimization for Multiple Analysis Jobs on MapReduce

Yoshifumi FUKUMOTO† and Makoto ONIZUKA†

† Nippon Telegraph and Telephone Corporation NTT Cyber Space Laboratories,
1-1 Hikarinooka Yokosuka-Shi Kanagawa 239–0847 Japan
E-mail: †{fukumoto.yoshifumi,onizuka.makoto}@lab.ntt.co.jp

1. はじめに

近年の記憶装置の発達やインターネットの普及により，企業が扱う情報の量が爆発的に増加しつつある．そしてその情報から統計解析やデータマイニングによって有益な知識が抽出され，有効活用されている．例えば Google・Yahoo!・facebook・eBay などでは膨大な量の Web データやログを処理し，スパム判定・文書分類・ユーザ行動分析などに実際に応用している．これらの企業を含む多くの企業では，大規模データ処理のために MapReduce [1] という実用的な分散処理技術が利用されており(注1)，MapReduce は大規模データ活用技術のデファクトスタンダードとして確立しつつある．大規模データを解析することで得られる知識はアルゴリズムによって異なるが，特に機械学習処理は単純な解析では知りえない，データの傾向や特徴を抽出することが可能となる重要な技術であり，MapReduce を用いた大規模データに対する機械学習が実際に行われている(注2)．

(注1)：http://wiki.apache.org/hadoop/PoweredBy
(注2)：https://cwiki.apache.org/confluence/display/MAHOUT/Powered+By+Mahout

そこで本研究では，アプリケーションとして機械学習を想定し，機械学習処理の特徴に基づいた MapReduce の改善を行う．

機械学習アルゴリズムに入力するデータは規模が大きいほど良い性能の結果が得られるが，一方で一定期間のうちに蓄積されるデータを次々と処理・活用するために，現実的な時間で処理を終えたいという要求がある．しかし，有用な機械学習アルゴリズムの中には，事前に与えるパラメータ（ハイパーパラメータ）値の処理結果に与える影響が大きいものがあり，パラメータ値最適化のためにパラメータ値を調整しながら処理を長時間に渡り何度も繰り返し行う必要があるため，非常に処理コストが大きくなってしまう．つまり機械学習による大規模データの有効活用には，機械学習アルゴリズムのハイパーパラメータ値最適化に伴う繰り返し処理の効率化が課題となっている．

本研究では，MapReduce を利用した機械学習の，ハイパーパラメータ値調整に依る複数回の処理において，入力データ・アルゴリズムが共通で，パラメータ値のみが異なる場合に，各処理間にパラメータ値に依存しない重複する部分があることに着目した．このような複数の処理の重複部分を検出し，それを共有化することにより処理効率を向上させる手法はマルチクエリ最適化 [9] と呼ばれる．MapReduce における自動マルチク

エリ最適化は Cheetah [10] で実現されている．Cheetah では MapReduce ジョブを SQL に似た抽象的・宣言的な言語で記述することを前提としており，既存の MapReduce 上で動作する機械学習アルゴリズムは Java による抽象化されていない手続き的な記述のものが大半であるため適用が難しい．MapReduce を利用した処理の記述が Java のままでもマルチクエリ最適化を実現する研究として，MRShare [11] が挙げられる．MRShare では入力データの読み込み部分を共有化し，さらに MapReduce 処理の途中で生成される中間データの冗長部分を削減することで処理を効率化しているが，機械学習アルゴリズムに対して適用する場合，中間データを削減できるケースが希少であるため効果が限定的になる．以上より，MapReduce における機械学習を考慮した自動的なマルチクエリ最適化が課題である．

そこで本研究では，パラメータ値のみが異なる複数の MapReduce ジョブが与えられたとき，ユーザに一切の負担を与えることなく自動的に複数のジョブを統合し，重複する処理部分を共有化して実行する手法を提案する．提案手法は事前に与えるパラメータの影響範囲を特定するパスと，それに基づいて複数 MapReduce ジョブを統合しパラメータに依存しない範囲を共有して実行するパスの 2 パスで構成される．1 パス目では十分に小規模なデータセットを用いて MapReduce ジョブを仮実行し，一元管理されているパラメータから Map フェーズ・Reduce フェーズで利用されるパラメータをそれぞれ抽出する．2 パス目では大規模な入力データを用いた複数の MapReduce ジョブを一度に実行する．開始時に各ジョブのパラメータを合成した上で，1 パス目で得た情報を用いて複数ジョブ間のパラメータに依存しない重複部分特定し，Map フェーズ・Reduce フェーズの単位で共有化する．ただし，Map フェーズが共有可能である場合 Map フェーズから出力される中間データも各ジョブ間で共有可能だが，中間データを共有すると後に続く Reduce 処理の粒度が大きくなるため，アルゴリズムや入力するデータによって負荷が偏ることがある．これを回避するために，Map フェーズまでを共有するものと Map フェーズ・Shuffle までを共有するもの 2 パターンを選択可能とする．さらに Map フェーズと Reduce フェーズが両方とも共有化可能である場合は MapReduce のジョブ単位での共有化を行う．

提案手法を実装し，機械学習ライブラリ Mahout の SVM アルゴリズムに適用して評価したところ，Map フェーズが重複範囲として共有化され，処理時間全体に対して Map フェーズが占める割合に応じて，Shuffle 処理までを共有化する場合は最大で 56 %改善した．また，共有化することによって Reduce 処理の粒度が大きなり，逆効果となる状況も実験によって示した．さらに，複数回の MapReduce ジョブから構成される NaiveBayes アルゴリズムに適用するとき，4 回の MapReduce ジョブのうち 3 回目までのジョブが共有可能であるため，提案手法が Map・Reduce フェーズ単位でなくジョブ単位の共有化に対応した場合，3 倍の高速化ができる可能性を実験により机上で確認した．

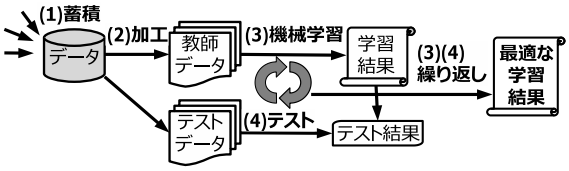

図 1　典型的な機械学習の流れ

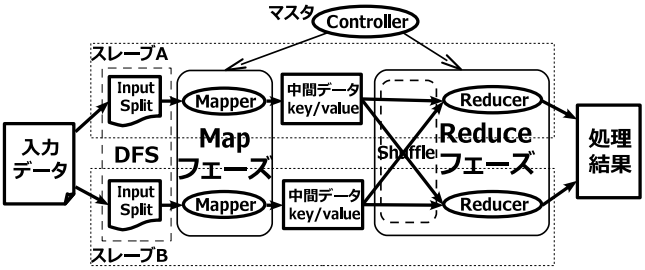

図 2　MapReduce の流れ

2. 予 備 知 識

2.1 機 械 学 習

典型的な機械学習は図 1 のようなフローで行われる．まず学習の対象となるデータを蓄積し (1)，次にそのデータをある要素を複数の属性（特徴）で説明するテーブル形式のデータの集合（1 属性 1 カラム (列)，1 要素 1 レコード (行)）である教師データに加工する (2)．さらにその一部をサンプリングしてテストデータとする．そして教師データを入力として機械学習を行う (3)．機械学習では，各属性のうちの一つをクラス属性とし，クラス属性以外の属性値を元にクラス属性値を決定する規則を学習処理によって出力する．出力された規則をテストデータに適用 (4) し，各属性から決定したクラス属性値がどれだけ正解のクラス属性値と一致しているかを調べることで，その規則の有用性を確認することができる．テスト結果が不十分である場合は，機械学習アルゴリズムに事前に与えるパラメータ値を変更して再度学習処理を行い新たな規則を得る．機械学習処理とテストを繰り返すうちに，最終的に最適な性能を持つ規則を得ることができ，この規則をクラス属性値が未知のレコードに対して適用することで活用する．

機械学習アルゴリズムにも依るが，事前に与えるパラメータの値によって，教師データに対してどれだけ忠実に学習するかを調節することができる．与えた教師データに特化して学習を行うと，そのデータに関しては限りなく精度の高い規則を得ることができるが，教師データ以外の未知のデータに適用したときの汎用性が損なわれる可能性が高くなる（過学習と呼ばれる）．教師データに対してどれだけ忠実に学習を行うかを調節するパラメータはハイパーパラメータと呼ばれ，適切な精度・汎用性を確保するにはハイパーパラメータの最適化を行う必要がある．

大規模なデータを機械学習処理の対象とした場合，入力データのバイアスによる影響が小さくなるため，小規模なデータには含まれない特徴・傾向が学習結果に反映できる可能性が高く

なる．つまり大規模であるほど，出力された規則の精度・汎用性を高めることができる．つまり学習対象とするデータは規模が大きいほど良いが，処理に要する時間が増大する．そのため，大規模データに対する機械学習の際に，学習アルゴリズムのハイパーパラメータ値の最適化は処理コストが非常に大きくなる．

2.2 MapReduce

本研究が改善の対象とする MapReduce について説明する．MapReduce ジョブは Map フェーズと Reduce フェーズの 2 フェーズから構成される．Map フェーズでは入力データ（D）を読み込んで Key/Value ペア（K_1, V_1）を生成し，それとパラメータ（P_m）を入力として各ペアに対してユーザが定義した Map 処理を行って，新たな Key/Value ペアリスト（$\{K_2, V_2\}$）を中間データとして出力する．Reduce フェーズでは，まず中間データを Key 部を元にグルーピング（Shuffle）する．そして各グループとパラメータ（P_r）を入力としてユーザが定義した Reduce 処理を行い，結果として Key/Value ペアリスト（$\{K_3, V_3\}$）を出力する．

$$
\begin{aligned}
D \rightarrow \quad & map(K_1, V_1, P_m) \rightarrow \{K_2, V_2\} \\
& shuffle(\{K_2, V_2\}) \rightarrow \{K_2, \{V_2\}\} \\
& reduce(K_2, \{V_2\}, P_r) \rightarrow (K_3, V_3)
\end{aligned}
$$

MapReduce 処理は複数のノード（コンピュータ）をネットワークで相互接続したクラスタ内で行われる．クラスタはマスタ（Controller）ノードと複数のスレーブノードで構成されており（図 2），クラスタ上に DFS（分散ファイルシステム）が構築され，入力データが格納される．MapReduce ジョブを開始すると，まずマスタが入力データをユーザの指定したサイズで分割し複数の InputSplit を生成する．次にマスタが InputSplit の数だけ Map タスクを生成し，各スレーブに（極力各々のローカルにある InputSplit が使われるように）割り当てる．Map タスクが割り当てられたスレーブでは Mapper が起動し，ユーザが定義した Map 処理が行われて，Key/Value 形式の中間データが出力される．中間データは同じ Key 値を持つものが一つのスレーブに集まるようにネットワークを介して相互に移動（Shuffle）される．マスタは中間データの Key 毎に Reduce タスクを生成し，各スレーブに割り当てる．Reduce タスクが割り当てられたスレーブでは Reducer が起動され，ユーザが定義した任意の Reduce 処理が行われて，新たに Key/Value 形式の処理結果が DFS に出力される．

3. 提 案 手 法

MapReduce を用いた機械学習のパラメータ値最適化に依る複数回の処理において，本研究では「アルゴリズム」と「入力データ」が共通で，事前に与える「パラメータの値」のみが異なるという点に着眼した．提案手法では，パラメータ値のみの異なる複数の MapReduce 処理を実行するとき，重複処理の自動的な共有化を実現する．

今回対象とする Java のような手続き型の言語で記述されている機械学習アルゴリズムの場合は，処理内容の事前把握や分解が難しいため，共有化可能な範囲を特定することも難しい．直感的な解としては，ソースコードの解析によって事前に与えたパラメータ値が影響する範囲を特定（プログラムスライシング）し，パラメータ非依存部分と依存部分に分離した上で，さらにパラメータに依存しない部分を，入力データやパラメータを特定の値に固定した状態でコンパイルし直し（部分評価），それを用いてプログラムを実行することが考えられる．しかし，ソースコードを入手可能であることが前提であり，実際にソースコードがあっても実現は難しいと考えられる．

そこで提案手法では，MapReduce の，パラメータ集合が処理から独立している点，Map フェーズ・Reduce フェーズのように処理が明確に分離されている点を活用して，Map フェーズ・Reduce フェーズといったフェーズを粒度とするパラメータ影響範囲の自動検出，処理共有化を行う．さらに，入力データ 1 スキャンの処理が MapReduce1 回に相当する点を活用し，MapReduce ジョブを粒度とした共有化も実現する．具体的には，1 パス目で MapReduce ジョブの仮実行を行い Mapper・Reducer がそれぞれ利用するパラメータを特定して共有範囲のヒントとする．2 パス目でパラメータ値が一部共通する複数のジョブを統合し，Map フェーズ・Reduce フェーズの単位で処理の共有化を行う．

3.1 パス 1：パラメータ値影響範囲特定パス

提案手法において解くべき問題は，2 つの MapReduce ジョブ（J_i, J_j）の重複する部分を抽出し，共有化しながら実行することである．本手法では，Mapper で利用されるパラメータ（P_m^i, P_m^j）と Reducer で利用されるパラメータ（P_r^i, P_r^j）を比較することで，Map フェーズ・Reduce フェーズの単位でジョブ間の重複部分の特定を可能である．

MapReduce のオープンソース実装である Hadoop では，各パラメータが MapReduce 中のどこで利用されるかは明示されない．そのため，パラメータに依存しない，またはパラメータ値が共通である複数の MapReduce ジョブ間にある重複処理範囲を特定するには，一元管理されている全パラメータから Map フェーズ・Reduce フェーズで利用されるものをそれぞれ抽出する必要がある．1 パス目では小規模のサンプルデータを利用して処理を行い，その過程で MapReduce における Mapper・Reducer がパラメータを利用するとき，それを検知して利用された Mapper 名・Reducer 名・関数名・パラメータ名を保存する．これにより全パラメータ群（P）から，Map フェーズ・Reduce フェーズで使われるパラメータ（P_m, P_r）を区別できるようにする．

3.2 パス 2：複数 MapReduce 統合・実行パス

2 パス目では本来対象とする大規模な入力データを利用した学習処理を行う．パラメータ値のみの異なる 2 つの MapReduce ジョブ（J_i, J_j）が与えられたとき，まず 2 つのジョブのパラメータ集合（P_i, P_j）の積集合と対象差を区別した上で一つのパラメータ集合を作成する．次に，1 パス目で得た各パラメータが Mapper・Reducer のどちらで利用されるかの情報（P_m, P_r）を参照し，対象差に含まれるパラメータがどのフェーズで利用されたのかを特定する（Algorithm1-1,2）．そして Map フェーズで利用する差分パラメータが無いジョブの Mapper をグルー

ピング（Algorithm1-3,4）する．Reduce フェーズは，パラメータ値が共通かつ Map フェーズでも同じグループに属していたジョブをグルーピングする（Algorithm1-6,7）．Map フェーズも Reduce フェーズも同じグループに属するジョブはジョブ単位での共有化が可能である（Algorithm1-9,10）．共有化できない場合は以降の処理をジョブ毎に個別に行う．

Algorithm 1 共有範囲の判定

Require: $J_i = \{M_i, R_i\}, J_j = \{M_j, R_j\}, P_i = \{p_1^i, p_2^i, \ldots, p_n^i\}$, $P_j = \{p_1^j, p_2^j, \ldots, p_n^j\}, P_m, P_r$

1: $P_m^{\Delta} = (P_i \Delta P_j) \cap P_m$
2: $P_r^{\Delta} = (P_i \Delta P_j) \cap P_r$
3: **if** $P_m^{\Delta} \equiv \phi$ **then**
4: $\quad M = \text{group}(M_i, M_j)$
5: **end if**
6: **if** $P_r^{\Delta} \equiv \phi, \{M_i, M_j\} \in M$ **then**
7: $\quad R = \text{group}(R_i, R_j)$
8: **end if**
9: **if** $\{M_i, M_j\} \in M, \{R_i, R_j\} \in R$ **then**
10: $\quad J = \text{group}(J_i, J_j)$
11: **end if**

本手法では学習アルゴリズムや与えるパラメータ，入力データなどの性質に応じて，a) 複数ジョブ間で Mapper を一切共有化できない場合 b) 中間データを共有すると負荷が偏る場合 c) 中間データを共有しても負荷が十分に分散される場合 d)Reducer まで共有化できる場合，それぞれに応じて共有化する処理の範囲を切り替えることで対応する．

a） データ読み込み部分のみ共有化

Mapper を一切共有化（グルーピング）できない場合（$P_m^i \neq P_m^j$）はワーストケースとなり，データ読み込み部分（$D \rightarrow \{K_1, V_1\}$）のみを共有化する．Map 処理結果として出力する中間データの Key 値にタグを付与することで，どのジョブのものかを区別する（$tag_i(K_2^i), V_2^i$）．

$$D \rightarrow \frac{map(K_1, V_1, P_i) \rightarrow \{tag_i(K_2^i), V_2^i\}}{map(K_1, V_1, P_j) \rightarrow \{tag_j(K_2^j), V_2^j\}} \frac{shuffle(\{tag_i(K_2^i), V_2^i\}) \rightarrow \{tag_i(K_2^i), \{V_2^i\}\}}{shuffle(\{tag_j(K_2^j), V_2^j\}) \rightarrow \{tag_j(K_2^j), \{V_2^j\}\}} \frac{reduce_i(tag_i(K_2^i), \{V_2^i\}, P_i) \rightarrow \{K_3^i, V_3^i\}}{reduce_j(tag_j(K_2^j), \{V_2^j\}, P_j) \rightarrow \{K_3^j, V_3^j\}}$$

b） Map を共有化

Mapper で利用するパラメータ値が共通（$P_m^{\Delta} = \phi$）で，かつ中間データの Key 値（K_2）のユニーク数がスレーブノードの数（N）よりも少ない場合（$|K_2| < N$）は，起動する Reducer がスレーブノードの数よりも少なくなってしまい，Reduce タスクが割り当てられないノードが発生して処理負荷が一部のノードに偏り，最大限の効率化ができない．そのためひとつの中間データに複数のタグを付与することで中間データの共有が可能でも，あえてジョブ別に中間データを出力して以降の処理はジョブ別に行う．つまり shuffle 処理はジョブ間で共有せず

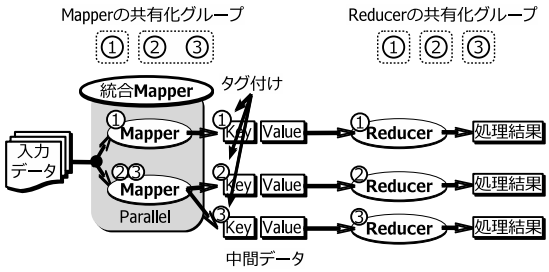

図 3　Map までを共有化（Reduce 処理粒度小）

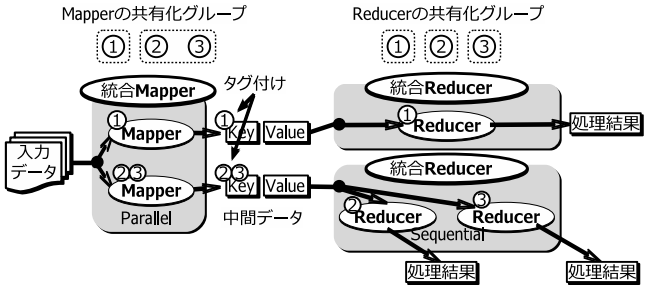

図 4　Shuffle までを共有化（Reduce 処理粒度大）

Reduce フェーズの負荷分散を優先する．

$$D \rightarrow map(K_1, V_1, P_i) \rightarrow \frac{\{tag_i(K_2^i), V_2^i\}}{\{tag_j(K_2^j), V_2^j\}} \frac{shuffle(\{tag_i(K_2^i), V_2^i\}) \rightarrow \{tag_i(K_2^i), \{V_2^i\}\}}{shuffle(\{tag_j(K_2^j), V_2^j\}) \rightarrow \{tag_j(K_2^j), \{V_2^j\}\}} \frac{reduce(tag_i(K_2^i), \{V_2^i\}) \rightarrow \{K_3^i, V_3^i\}}{reduce(tag_j(K_2^j), \{V_2^j\}) \rightarrow \{K_3^j, V_3^j\}}$$

図 3 はパラメータ値の異なる 3 つのジョブがあったとき，2 番目と 3 番目のジョブの Map フェーズを共有化する場合の処理の流れを示している．

c） Map・Shuffle を共有化

Key 値のユニーク数が十分に多い場合（$P_m^i = P_m^j, |K_2| \geq N$）は中間データを共有しても負荷が分散されるため，共有化された中間データを Reducer に入力し，ひとつの Reducer で複数ジョブ分の Reduce 処理を実行する．

$$D \rightarrow map(K_1, V_1, P_i) \rightarrow \{tag_{ij}(K_2), V_2\} \quad shuffle(\{tag_{ij}(K_2), V_2\}) \rightarrow \{tag_{ij}(K_2), \{V_2\}\} \quad \frac{reduce(tag_{ij}(K_2), \{V_2\}, P_i) \rightarrow \{K_3^i, V_3^i\}}{reduce(tag_{ij}(K_2), \{V_2\}, P_j) \rightarrow \{K_3^j, V_3^j\}}$$

図 4 は 2 番目と 3 番目のジョブの Map フェーズと Shuffle を共有化する場合の処理の流れを示している．

d） Map・Shuffle・Reduce を共有化

Mapper だけでなく，Reducer で利用するパラメータ値も全て共通ないしパラメータに依存しない場合は，MapReduce ジョブ単位での共有化となる．

$$D \rightarrow map(K_1, V_1, P_i) \rightarrow \{tag_{ij}(K_2), V_2\} \quad shuffle(\{tag_{ij}(K_2), V_2\}) \rightarrow \{tag_{ij}(K_2), \{V_2\}\} \quad reduce(tag_{ij}(K_2), \{V_2\}, P_i) \rightarrow \{K_3, V_3\}$$

4. 評価・考察

MapReduce のフェーズ単位での共有化の効果を確認するために，提案手法を実装して SVM アルゴリズムに適用し，一度に 10 パターンのパラメータ値を使った処理を実行する実験を行った．また，MapReduce のジョブ単位での共有化の効果を確認するために，10 パターンのパラメータを用いた NaiveBayes アルゴリズムを一度に実行し，詳細な処理時間を測定した．

4.1 MapReduce のフェーズ単位での共有化の効果

4.1.1 実験環境・実験条件

分散処理用のクラスタとして，Core2Duo1.86GHz，8GB RAM のサーバ 10 台を 1Gbps のネットワークで相互に接続した環境で実験を行った．各サーバにはオープンソースソフトウェア Hadoop のバリエーションである，Cloudera's Distribution for Hadoop 3（CDH3）(注3) を導入した．Hadoop のチューニングはソート用のメモリ設定を 256MB にし，今回実験に利用するアルゴリズムは Reducer が多くのメモリを消費するため 1 台で動作する Mapper・Reducer の数を一つに設定してそれぞれ 4GB までメモリを利用できるようにした．その他の Hadoop の設定は初期状態と同じである．

機械学習処理は，Hadoop 上で動作する機械学習ライブラリ Mahout(注4) を導入することで実現した．実験には Mahout のパッチとして提供されている SVM アルゴリズム(注5) を利用し，実験と関係の無いアルゴリズムのパラメータは実験環境に対して極力の最適化を行った．SVM は特に過学習の程度を調整するハイパーパラメータ値の調整が難しいアルゴリズムである．今回利用した SVM アルゴリズムの Map フェーズは入力データのサンプリングと複数ノードへの配置を行い，実質的な学習処理が Reduce フェーズで行われるため，Map フェーズは完全に共有化が可能となる．Reduce ではハイパーパラメータが利用されており，今回はこのパラメータの値を複数パターンに振らせて実験を行った．

教師データとして入力するデータは LIBSVM の多クラス分類を行うためのデータセット集から最もサイズの大きいもの(注6) を選択した．スパースベクタの形で保存された 810 万レコード・約 18GB のデータで，属性（特徴）は 784 種類あり，各レコードには分類先として 10 種類のクラスのうちどれか一つがラベルとして付加されている．今回の評価実験ではこの教師データを用いて，SVM アルゴリズムによる多クラス分類を行う．Mahout の SVM アルゴリズムでは Map 処理の結果として出力される中間データ（Key/Value）の Key 値は教師データのラベルとなるため，Key 値のユニーク数はクラス数と同じ 10 となる．つまり 1 ジョブにつき Reducer は 10 個起動することになる．

実験は共有化による処理量削減効果の確認・共有処理時間の割合が効果に及ぼす影響の確認・中間データ共有化によって Reduce 処理の粒度が大きくなる弊害の確認を目的とし，SVM のハイパーパラメータである Lambda 値（過学習の程度を調整するためのパラメータ）が異なる 10 パターンの MapReduce ジョブを Hadoop に同時に与え，共有化しない場合と，共有化する場合の処理時間を測定した．

(注3)：http://www.cloudera.com/hadoop/
(注4)：http://mahout.apache.org/
(注5)：https://issues.apache.org/jira/browse/MAHOUT-232
(注6)：http://www.csie.ntu.edu.tw/˜cjlin/libsvmtools/datasets/ multiclass.html#mnist8m

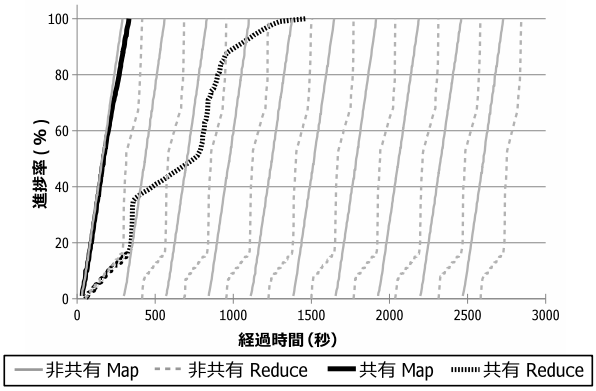

図 5　非共有と共有の比較

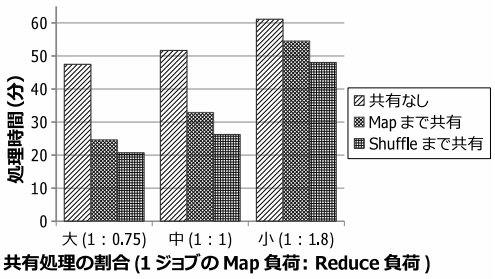

図 6　全処理時間中の共有処理の割合と共有範囲による影響

4.1.2 共有化による効果

図 5 は Map までを共有化した場合と，共有化しない場合を比較した経過時間に対する各ジョブの進捗状況を示すグラフである．グラフの横軸は経過時間，縦軸は Hadoop のログとして出力される 1 ジョブの Mapper・Reducer の進捗率を示している．共有化しない場合は 10 パターンのジョブが Mapper・Reducer のグラフ（灰色）各 10 本ずつ現れ，一方共有化した場合は 10 パターン全てが 1 ジョブに統合されるため，Mapper・Reducer のグラフ各 1 本ずつ（黒色）現れている．グラフは 1 ジョブだけ実行した場合の Map 処理時間と Reduce 処理時間の比率が 1 : 0.75 の場合のものである．

共有化により約 56 %の改善となった．実際に図 5 の共有と非共有のグラフを比較すると，共有 Map は非共有 Map1 つ分＋ α の処理時間を要して 1 回で終了しており，Map 処理 9 回分の削減が高速化に最も貢献していることが分かる．

Map フェーズを完全に共有化できるため，理想的には処理時間全体で 50 %以上（処理負荷 : $10Map + 7.5Reduce = 17.5 \rightarrow 1Map + 7.5Reduce = 8.5$）の改善が見込まれており，データ読み込み部分の共有化による処理量の削減や，Map のみ共有化する場合では共有 Mapper が出力する総中間データサイズが非共有と同等（10 ジョブ分＋タグの分）となるため共有 Map

フェーズの負荷が非共有時の 1 回分よりも大きくなったこと，それにともなって Shuffle コストが増加したこと，その他各所のオーバーヘッドが含まれている．

4.1.3 共有処理時間の割合による影響

図 6 は入力データをサンプリングするパラメータ値を変更することによって，Map フェーズの負荷を固定したまま Reduce フェーズの負荷を変動させ，1 ジョブを実行するときの Map フェーズ負荷：Reduce フェーズ負荷を図で示した通りに調整して，それぞれにおいて共有化なし，共有範囲が Map まで，Map・Shuffle までの 3 パターンを試行した場合の処理時間を示している．

結果は図 6 のようになった．共有化できる Map 処理時間の割合が大きいほど提案手法の効果が大きくなっているが，これは共有化可能な Map フェーズに対して Reduce フェーズ負荷が小さくなるほど，共有化部分の処理時間全体に対する割合が高くなるからであると考えられる．また，共有範囲が広い Shuffle までの共有化のほうが効果が大きくなっている．Shuffle まで共有化する場合は Map まで共有化する場合と比較して出力される中間データが 1 ジョブ分（10 分の 1）に抑えられ，さらに Reduce フェーズにおける中間データの読み込み部分も 10 ジョブ間で共有されたため，このような結果になったと考えられる．

4.1.4 共有化の弊害

今回の実験データの性質上，スレーブノードが 10 台以上の環境で実行する場合は Redue タスクが割り当てられるノードが全ノード数よりも少なくなってしまい，効率が悪くなることが予想されたため，あえてその影響が発生すると考えられる 20 台での実験を行った．

その結果，Shuffle までを共有化した場合，スレーブノード数が 20 台の環境では逆に共有化を行わない場合よりも処理時間が悪化した．これは想定していた通り処理時間の後半は Reduce タスクが割り当てられていないスレーブノード 10 台がアイドル状態となっていたことが原因であると考えられる．つまり Shuffle までの共有化を行う場合は Reduce 処理の粒度が大きくなり，中間データの Key 値のユニーク数によっては逆に効率が悪化するため，Map までの共有化に留めるべきである．

4.2 MapReduce ジョブ単位での共有化の検討

4.2.1 実験環境・実験条件

SVM と同スペックのサーバ 20 台にて実験を行った．Mapper・Reducer は 1 ノードあたり 2 個まで起動可能にし，それぞれ 2GB までメモリを利用できるようにした．その他の Hadoop の設定は SVM の実験と同じとした．

機械学習アルゴリズムとして，NaiveBayes を利用した．Mahout の NaiveBayes では 1 度の学習処理の中で 4 回の MapReduce ジョブが実行される．このうち 1 回目から 3 回目の MapReduce ジョブは調整の必要なパラメータに依存していないため，ジョブ単位での共有化が可能であることが見込まれた．実験は Mahout の Web サイトのサンプル(注7)を参考に，学習の平滑化に関わるパラメータ値（AlphaI）を 10 パターン用意して同時に実行した．今回は Reduce フェーズの共有化を含む MapReduce のジョブ単位の共有化を行うための実装を用意できていないため，通常の Hadoop を用いた非共有の実験のみ行い，提案手法によって共有化可能な範囲を全て仮に共有した場合の効果を推測することを目的とする．

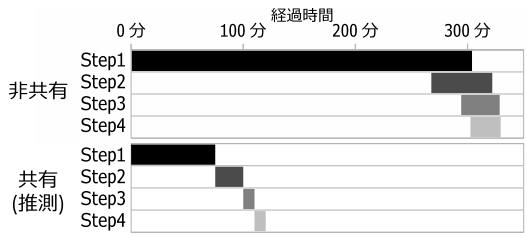

図 7 NaiveBayes10 回分の各 MapReduce ジョブの処理時間とジョブ単位共有化が実現した場合の推測処理時間

4.2.2 ジョブ単位での共有化の効果

NaiveBayes は MapReduce ジョブ 4 回（Step1~ Step4）から構成され，10 個のパラメータ値の NaiveBayes を処理する場合は MapReduce が計 40 回実行される．図 7 は 10 回分を同時に開始したとき，Step1~ Step4 のジョブがそれぞれ初めて開始された時刻とパラメータ値 10 個分全てのジョブがそれぞれ完了した時刻を示している．上部は通常の Hadoop を用いて処理した非共有時の実験結果で，下部はジョブ単位での共有化が実現された場合の推測値となっている．

結果より，特に Step1 の処理時間が最も長く，ジョブ単位で共有可能であると考えられる Step3 までに全処理時間の 9 割以上を費やしていることが分かった．また，あるパラメータ値を使った処理の Step 間に依存関係はあるが，別のパラメータ値を使った処理の Step とは依存関係がないため，図 5 のようにサーバクラスタのリソースの許す限り各 Step の MapReduce が重なりあうようにして実行されている．これを共有化し Step3 までをそれぞれ 1 回の MapReduce ジョブで処理する場合は，Step 間の依存関係があるため重なって処理されることは無いと考えられる．そのため，ジョブ単位の共有化によって単純に 10 分の 1 の時間に圧縮することはできない．実際の実験結果を元に，共有化した Step3 までの MapReduce ジョブが 1 回ずつ順次実行されるとすれば，図 7 の下側のように全体で 3 倍近い高速化が可能であると考えられる．共有化できない Step4 も非共有のものよりも短く推測しているのは，共有化した場合は Step4 を同時に並列に開始できるためである．

5. 関 連 研 究

本研究は大規模データ処理を分散処理技術によって高速化する，いわゆるクラウドコンピューティングと総称される分野に属する．さらにその中でも分散処理フレームワーク MapReduce を対象に従来手法よりも効率の良い処理を目指している．MapReduce を対象とする研究は，エンジン側である MapReduce そのものを改善する方向と，その上で動くアプリケーション側を改善する方向に分けられる．

(注7)：https://cwiki.apache.org/confluence/display/MAHOUT/Wikipedia+Bayes+Example

5.1 アプリケーション側の工夫

本研究の実験で Hadoop 上で動作するオープンソースの機械学習ライブラリ Mahout を用いたが，このプロジェクトの原点となる研究 [4] では代表的な機械学習アルゴリズム 10 種を MapReduce フレームワークを用いて分散処理できるよう改変し，アルゴリズム毎に特徴を整理した上で MapReduce で処理できる形に変更する指針を示している．その他にも既存の解析処理や機械学習アルゴリズムを MapReduce に最適化する研究が多数行われている．PEGASUS [2] では MapReduce を用いた行列処理の最適化を行い，Google の PLANET [3] では決定木生成処理を MapReduce に最適化することで処理を高速化している．

5.2 エンジン側の工夫

エンジン側の原点は Google の MapReduce である．MapReduce にデータベース分野の技術を持ち込むことにより，生産性の向上や処理の効率化を実現できる場合がある．Pig [5] や Hive [6] は MapReduce 処理を抽象化した言語で記述できるようにしており，MapReduce プログラミングの生産性を高めている．

機械学習などの分析処理を考慮して MapReduce を改善する技術として HaLoop [7] や Twistar [8] が挙げられる．PageRank 計算や K-means アルゴリズムでは複数回の MapReduce を実行して同じデータを何度も読み込むという性質があり，そのような処理を中間データや処理結果のキャッシュによって効率化している．このキャッシュ機能は機械学習アルゴリズムのパラメータ値最適化にも活用できる可能性がある．

5.3 マルチクエリ最適化技術

提案手法は，データベース分野における「マルチクエリ最適化」[9] の思想を MapReduce に持ち込むものであり，複数のタスクを処理するときに，個々のタスクを細分化・順序の変更をして処理やデータを共有することで，全体的な効率を向上させる技術である．SQL のような抽象的・宣言的な言語を用いて処理を記述する場合は事前に処理内容全体の把握が可能で，処理の分解・順序変更も可能であるため最適化が容易である．

Cheetah [10] は MapReduce 処理を SQL ライクな言語を用いて記述可能にしたもので，処理を分解し，MapReduce の性質に基づいた最適な順序に組み替えてから実行する機能を提供している．しかし，機械学習は処理を SQL で記述可能かどうかはアルゴリズムに依存しており，大半の MapReduce を利用した機械学習アルゴリズムは Java で記述されているため移行コストが大きい．MRShare [11] では，Java 記述のままでのジョブ共有化を実現している．事前処理として複数のジョブをクラスタリング・統合し，入力データの読み込み部分を共有化して個々のジョブを実行する．さらに出力された中間データのうち Value 値が共通のものを共有化し，データサイズを削減することで Shuffle コストや Reduce 時のデータ読み込みコストを低減している．データ読み込み部分共有化はジョブの入力データが共通であれば一定の効果を発揮するが，機械学習の場合は中間データの Value 値が共通になることは希少であり，効果が限定される可能性が高いため適さない．

6. おわりに

大規模な機械学習処理を MapReduce による分散処理を利用して行う場合に，機械学習アルゴリズムのハイパーパラメータ値最適化に伴う繰り返し処理が高コストであるという問題に対して，複数回の MapReduce 処理をひとつに統合し，さらにパラメータに依存しない重複する処理を共有化することによって処理量を削減する手法を提案した．

提案手法を SVM アルゴリズムに適用して評価実験を行い，10 パターンの MapReduce ジョブを与えられたとき Map フェーズのみを共有化する場合に最大で 48 %，Shuffle までを共有化する場合に最大で 56 %処理時間が改善することを確認した．また Shuffle までの共有化よりも Map までの共有化に留めた方が良い状況について示した．

また，評価実験により提案手法を 4 つの MapReduce ジョブで構成される NaiveBayes アルゴリズムに仮に適用できた場合，10 パターンのパラメータ値を用いた学習処理を行う際に 3 つ目のジョブまでを共有化することによって，約 3 倍高速化できる可能性を示した．

今後の課題として以下の 3 つが挙げられる．

a） 共有化による弊害と負荷分散

図 6 の評価実験のノード数 20 台で Map・Shuffle を共有化した場合において，特に共有処理部分の全処理時間に対する割合が少ないときに提案手法が逆効果となっていた．これは共有化によって処理の粒度が大きくなり，負荷が偏ることが原因となっている．

今回の実装では，Map まで共有化・Shuffle まで共有化のどちらかを事前に指定する形にしていたが，自動的に有効な方を選択するべきである．中間データの Key 値のユニーク数とスレーブノードの数が判明すれば自動化することが可能だが，Key 値のユニーク数は完全にアルゴリズム依存かつデータ依存となっており，事前に把握することが難しい．また，Key 値やパラメータ値によって個々の Reduce 処理の負荷が大きく変わるようなアルゴリズムである場合には対応できない．この問題を解決するには処理中に何らかの形で Map 負荷や Reduce 負荷を予測し，適切に負荷分散するように中間データを各 Reducer に配置できるようにすることが必要である．

b） 共有化粒度の柔軟性向上

提案手法では MapReduce が Map フェーズ・Reduce フェーズのように処理が明確に 2 分されている性質を利用して，それを共有範囲の単位として利用した．しかし Mapper・Reducer 内でパラメータが複数利用される場合や，Map フェーズ内の後半でパラメータが利用され，前半はパラメータに依存しない場合など，より小粒度で共有化を行うことによって効果が大きくなるケースも存在する可能性がある．

c） 共有化を前提とした MapReduce 処理の記述

今回の提案手法実現のための実装により，複数 MapReduce ジョブの共有化による効果を測ることが容易になった．今後は実験に利用したもの以外の機械学習アルゴリズムや，その他の解析処理に提案手法を適用してみて，それぞれの処理の特徴や

共有化の効果などを調査し，共有化しやすい処理の性質や記述方法を整理すべきである．

文　　献

[1] Jeffrey Dean, Sanjay Ghemawat, "MapReduce:Simplified Data Processing on Large Clusters", OSDI 2004.

[2] U Kang, Charalampos E Tsourakakis, Christos Faloutsos, "PEGASUS: A Peta-Scale Graph Mining System - Implementation and Observations", ICDM 2009.

[3] Biswanath Panda, Joshua S. Herbach, Sugato Basu, Roberto J. Bayardo, "PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce", VLDB 2009.

[4] Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, Yuan Yuan Yu, Gary Bradski, Andrew Y. Ng, Kunlie Olukotun, "Map-Reduce for Machine Learning on Multicore", NIPS 2006.

[5] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, Andrew Tomkins, "Pig latin: a not-so-foreign language for data processing", SIGMOD 2008.

[6] Facebook Data Infrastructure Team, "Hive - A Warehousing Solution Over a Map-Recduce Framework", VLDB 2009.

[7] Yingyi Bu, Bill Howe, Magdalena Balazinska, Michael D. Ernst, "HaLoop: Effcient Iterative Data Processing on Large Clusters", VLDB 2010.

[8] Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, Geoffrey Fox, "Twister: A Runtime for Iterative MapReduce ", HPDC 2010.

[9] Timos K Sellis, "Multiple-Query Optimization", ACM Transactions on Database Systems 1988.

[10] Songing Chen, "Cheetah: A High Performance, Custom Data Warehouse on Top of MapReduce", VLDB 2010.

[11] Tomasz Nykiel, Michalis Potamias, Chaitanya Mishra, George Kollios, Nick Koudas, "MRShare: Sharing Across Multiple Queries in MapReduce", VLDB 2010.