2017年12月1日　小野田成晃

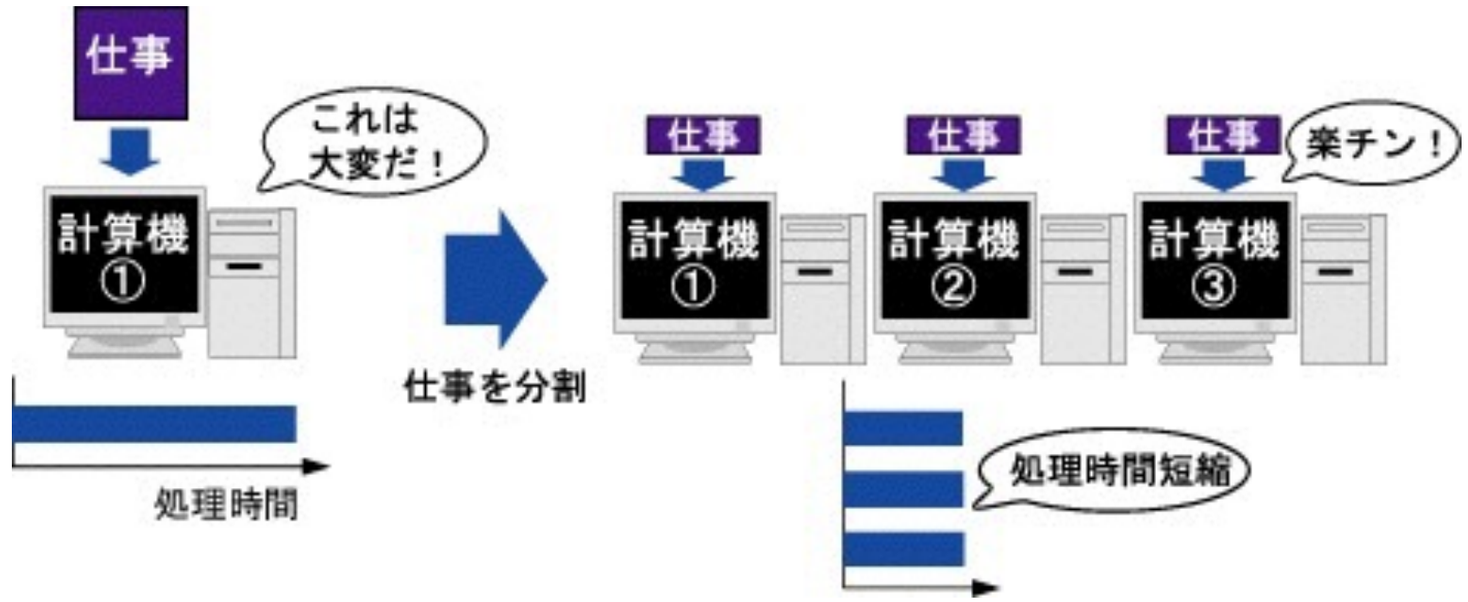
複数機械学習処理における MAPREDUCE 最適化

MapReduceとは

- 大規模データ分散バッチ処理（並列分散処理）を実現するフレームワーク
- テラバイト級の大規模なデータを処理することに用いられる。
- それをJavaで実装したものがHadoopと呼ばれる
- Javaの機械学習ライブラリMahout(Pythonで言うscikit-learn)と併用することで機械学習による高度な処理が可能
- Hadoop上でSQLライクにDBを操作するソフト:Hiveも有名
- →そもそも並列分散処理とは？

並列分散処理とは

- 一台のコンピュータで解かせることが困難な問題を複数のコンピュータに配分して解かせることで、処理時間が高速になる

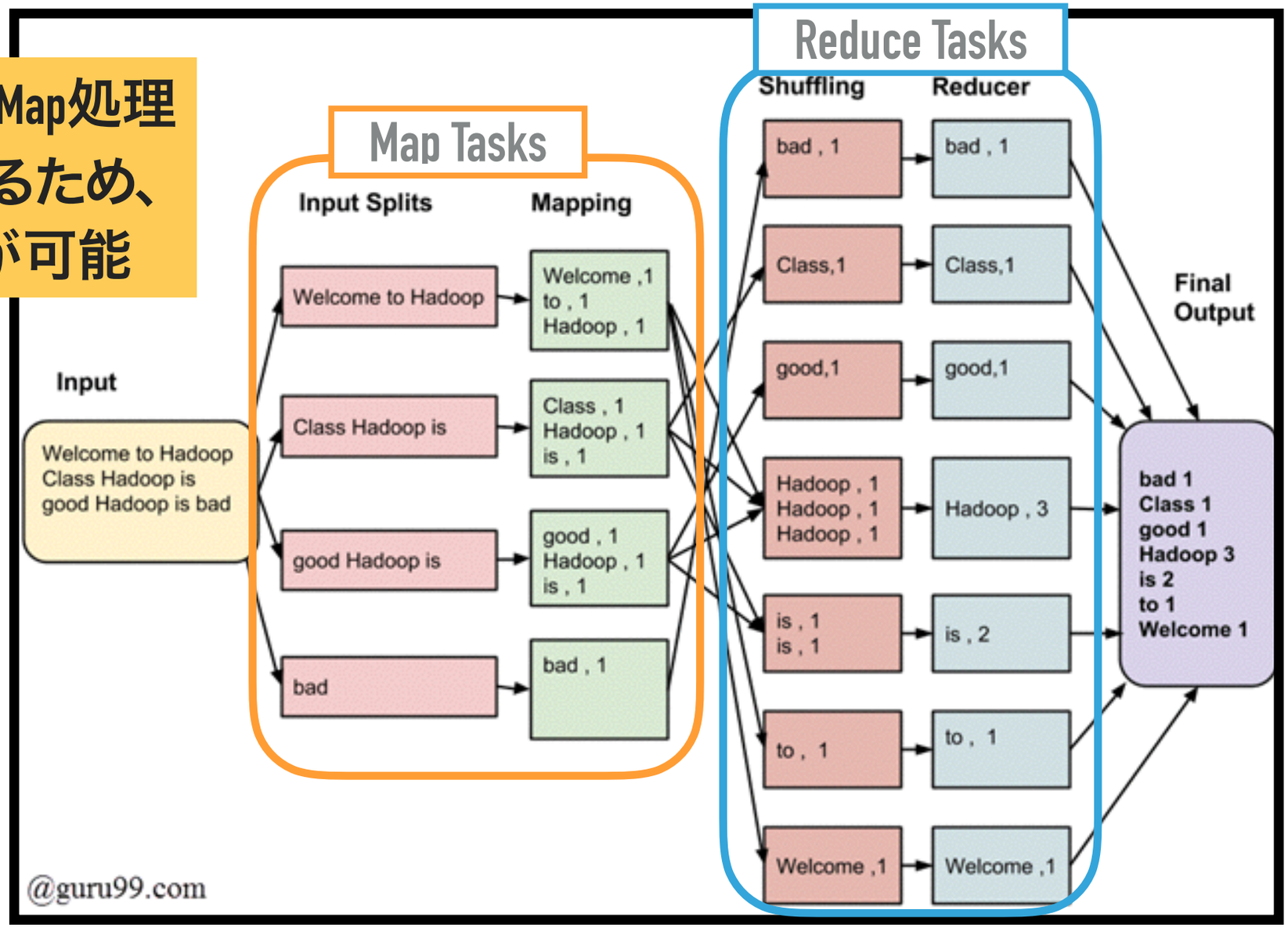


- しかし、実際には異なるPCではミドルウェア、ディレクトリ構成が異なることから、分散処理は容易ではない

MapReduceにおいての並列分散処理

- 1.Map tasks (Spilts & Mapping) 2.Reduce tasks (Shuffling, Reducing)

各Map処理は他のMap処理と完全に独立であるため、全て並列実行が可能



フレームワークの登場

- そこで、ハードウェアなど分散処理に必要な細かな知識がなくても、より簡単に並列分散処理を実現できる仕組みがMapReduceなどの分散処理フレームワークである！

- 主な分散処理フレームワーク一覧

- 1.Hadoop/MapReduce: 処理をMap・Reduceに分けて実行
- 2.Spark: 高速なフレームワーク、Hadoopと共存が可能
- 3.Jubatus: 日本製、オンライン機械学習に特化←卒研で使う予定

MapReduceの問題点

- ハイパラメータ：SVM等の機械学習では入力するパラメータによって、処理結果の品質が大きく変わるようなパラメータのこと
- MapReduceでは、このハイパラメータの最適化に対しての透過性が不足している
- そこで新たなハイパラメータ最適化手法を考案

先行研究・技術

	Haloop	Twister	Incoop	MRshare
透過性	×	×	×	△
理由	独自APIに従ってアルゴリズムを記述して再利用を定義する必要あり		メモ化データ再利用可否判断についてはユーザが判断する必要あり	機械学習処理の総処理量を削減可能だが自動判定がないためユーザの知識依存になる

- どの技術もユーザが静的に共有部分を決める必要あるため、透過性に難あり

前提条件・設計

- Hadoop上で動作
- 機械学習にはMahoutを想定
- 機械学習アルゴリズムをブラックボックスにして、表層的に得られる情報のみで自動的な共有可能処理を判定
- 共有部分を検出→共有可能部分を一つのジョブにマージ

フレームワークの適用例

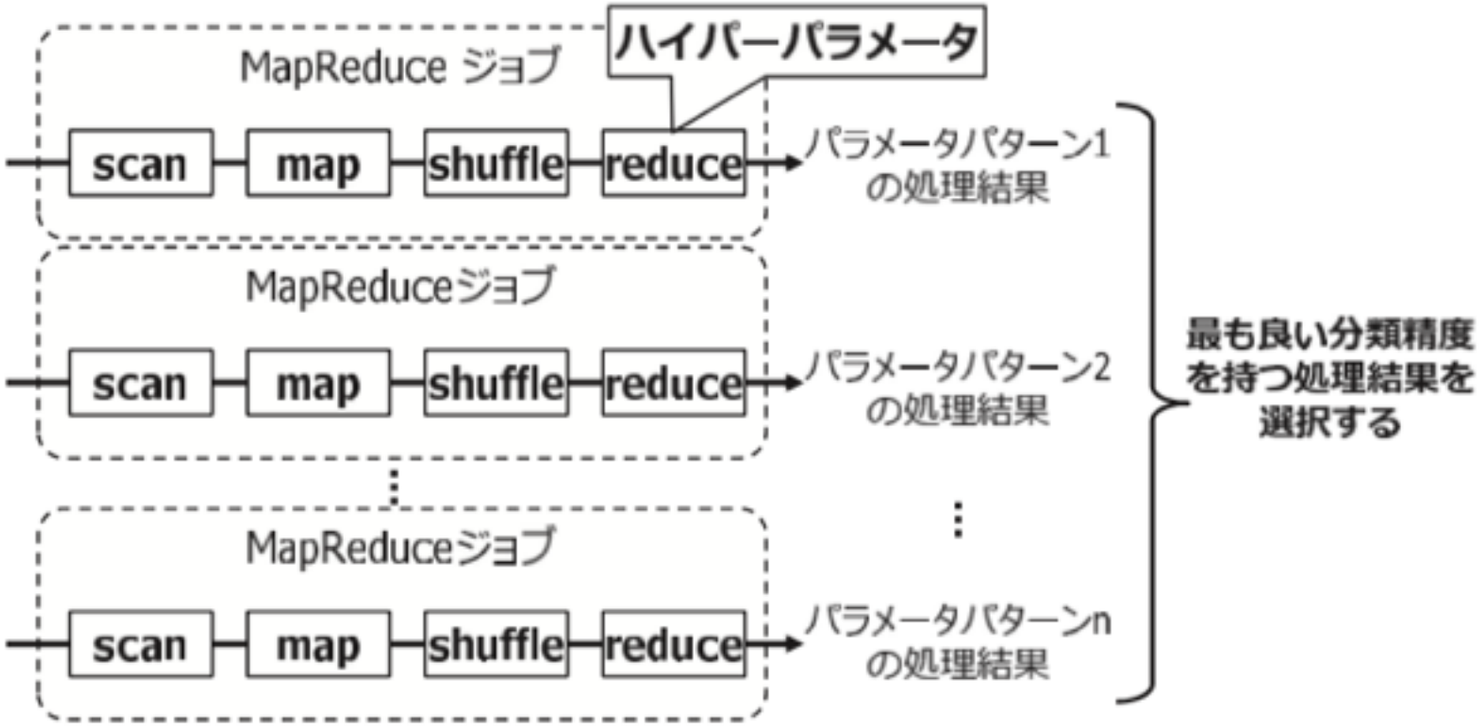


図 3　SVM の複数処理列

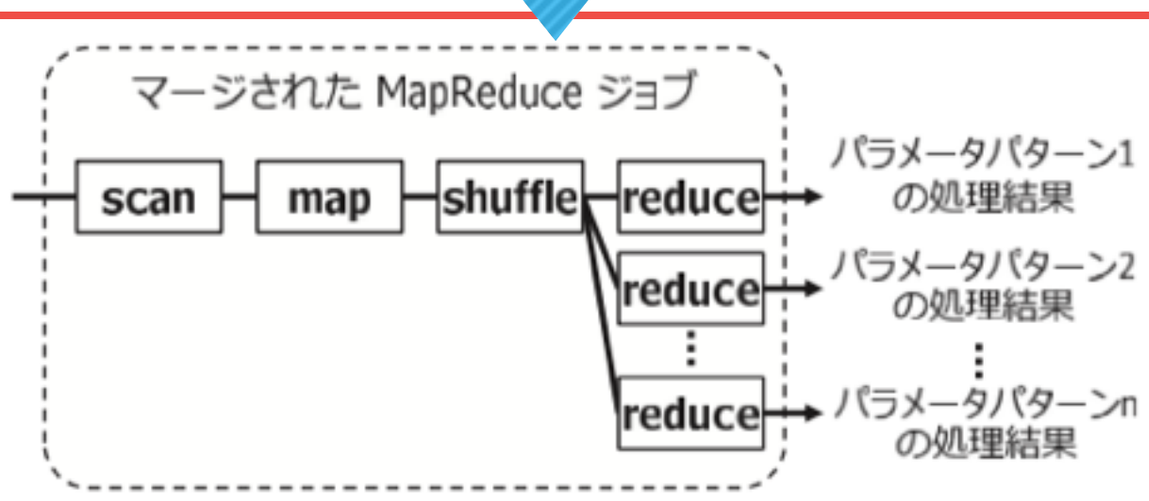
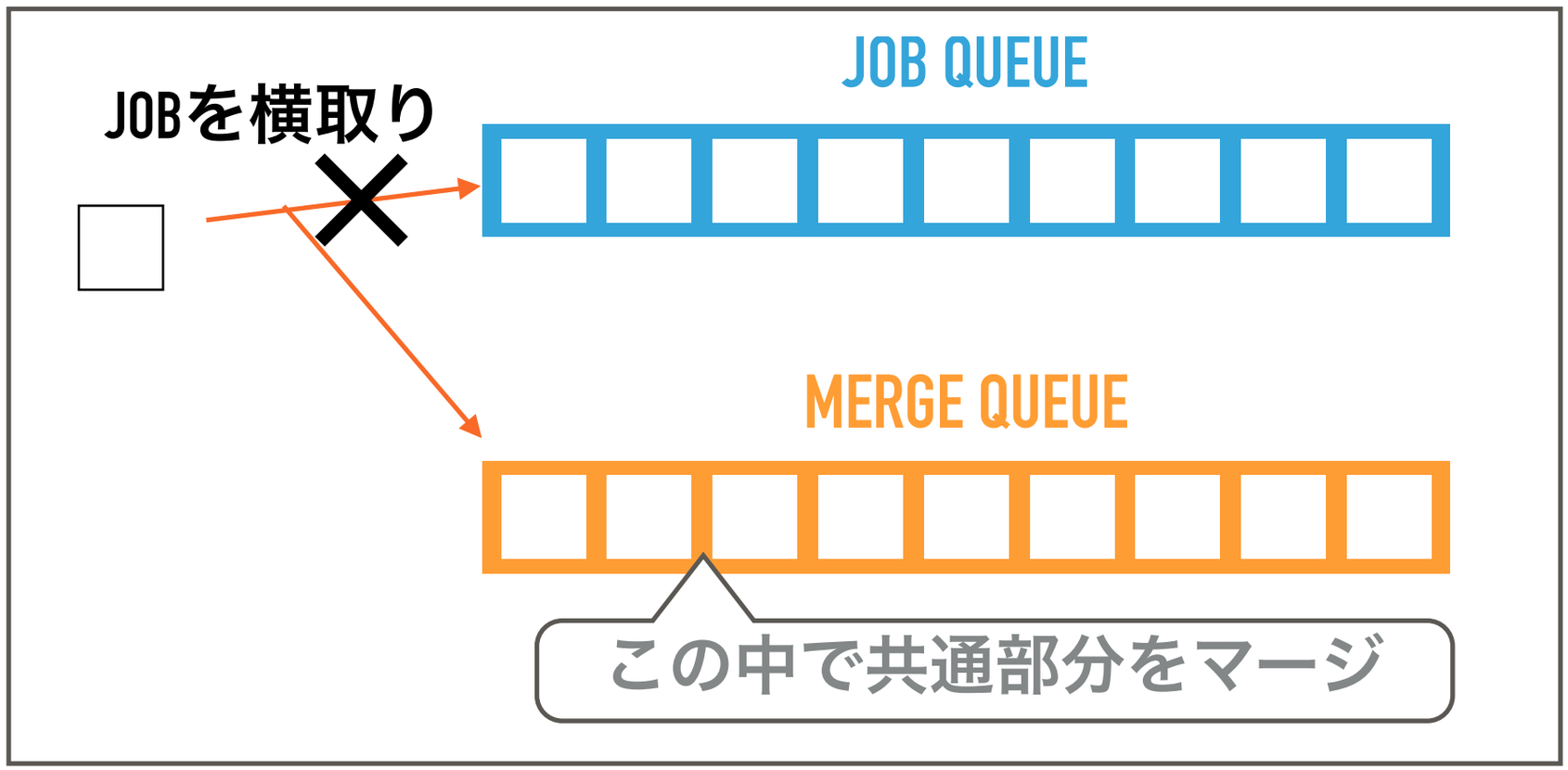


図 4　SVM の共有実行プラン

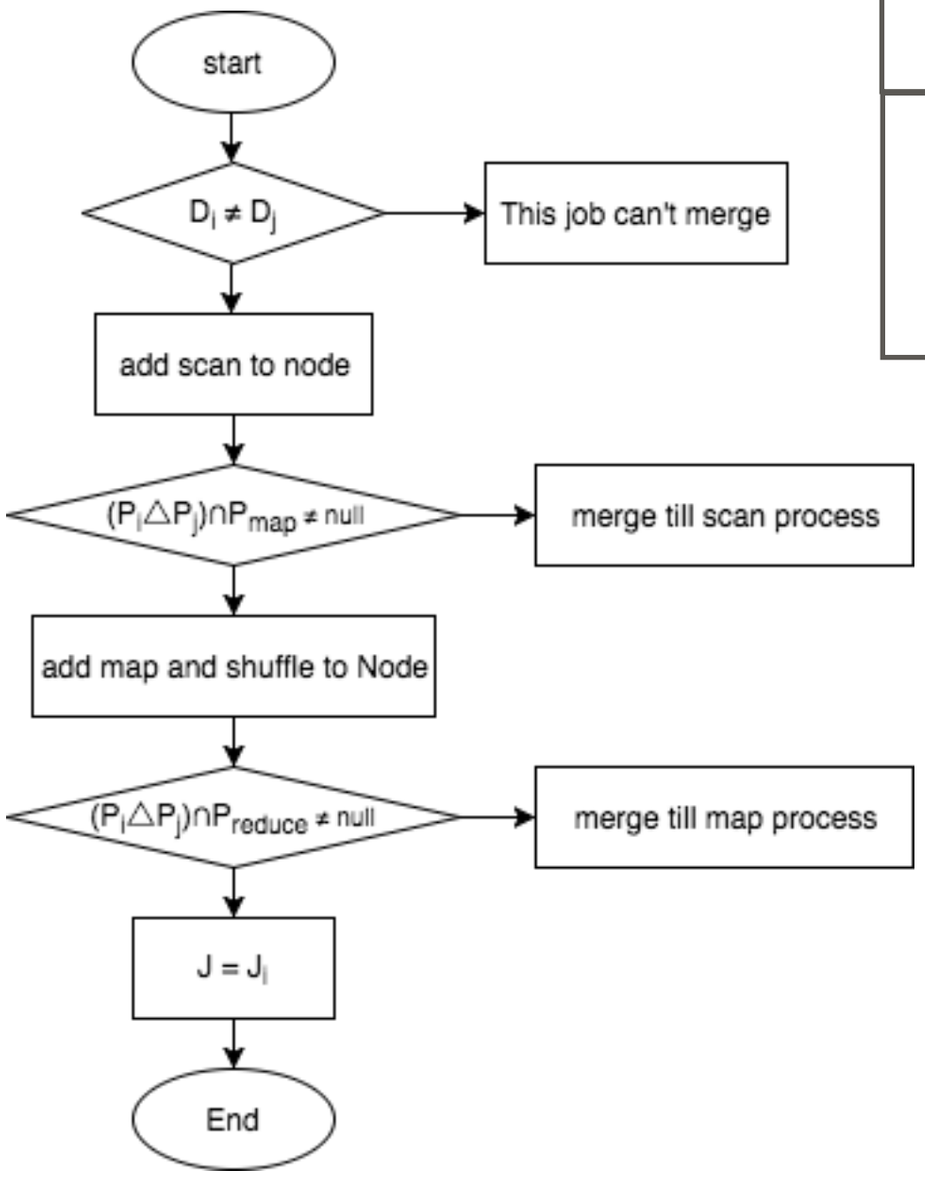
- SVMを分析例としてフレームワークを利用
- 1.複数のハイパラメータの組み合わせで実行
- 2.reduceタスクのみハイパラメータが影響していると判明
- 3.共有化できる部分をマージ
- 4.図４のようなジョブになる

共有化アルゴリズム-共有実行プラン



- パラメータのみ異なる複数処理列が開始されたとき、Hadoopは MapReduceジョブを一旦JobQueueに投入する
- 本研究のフレームワーク、そのジョブを横取りしてMergeQueueに投入、ジョブの開始時間を指定時間遅延させる
- その間にマージ可能なジョブをマージする

共有化アルゴリズム-ジョブのマージ



前提

$$Input : J_i, J_j, D_i, D_j, P_i, P_j, \{P_{map}, P_{reduce}\}$$

$$J = newMergeJob()$$

前ページのマージ部分のアルゴリズム

このアルゴリズムでは共有できない処理単位があった場合、
その時点で判定を終了して、それ以降の処理単位は全て共有不可とする

共有化されたジョブの実行パターン

- Hadoop上で動作
- 機械学習にはMahoutを想定
- 機械学習アルゴリズムをブラックボックスにして、表層的に得られる情報のみで自動的な共有可能処理を判定
- 共有部分を検出→共有可能部分を一つのジョブにマージ

評価実験

実験環境

- CDH3を拡張する方法でフレームワーク実装
- 10台ないし20台の物理サーバマシンを利用
- 機械学習であるLIB-SVMを対象として実行

三種の共有化効果検証

- 1. 処理単位の共有化効果
- 2. ジョブの共有化効果
- 3. 反復タイプアルゴリズムの共有化効果

処理単位の共有化効果

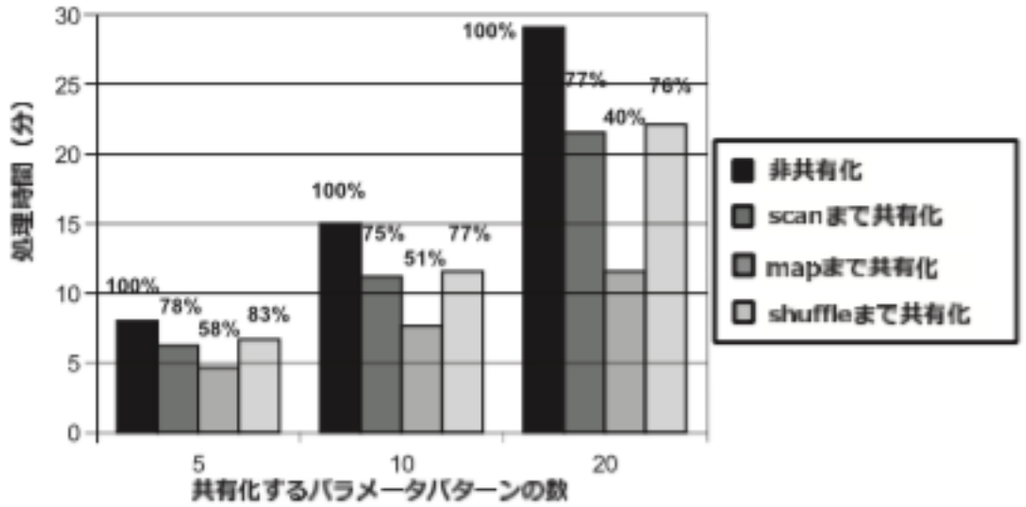


図 10　各処理単位の共有化効果（サーバ 20 台）

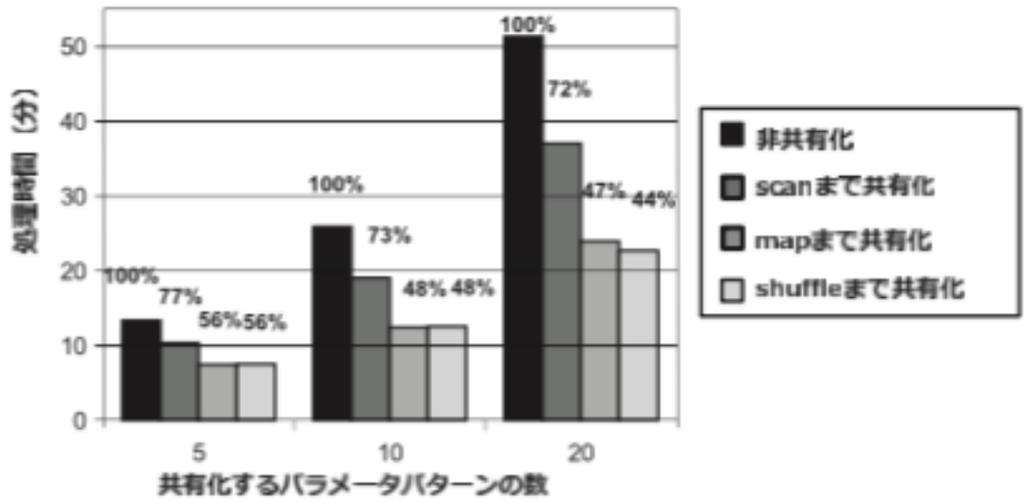


図 11　各処理単位の共有化効果（サーバ 10 台）

- 基本的に共有化した方が処理時間は短縮できている
- shuffle処理のみ手書き数字が0~9の10種であるため並列度が不足して処理時間が短縮できていない
- 20台より10台のほうが効果が顕著

ジョブの共有化効果

縦軸はハイパラメータのパターン数（１０）

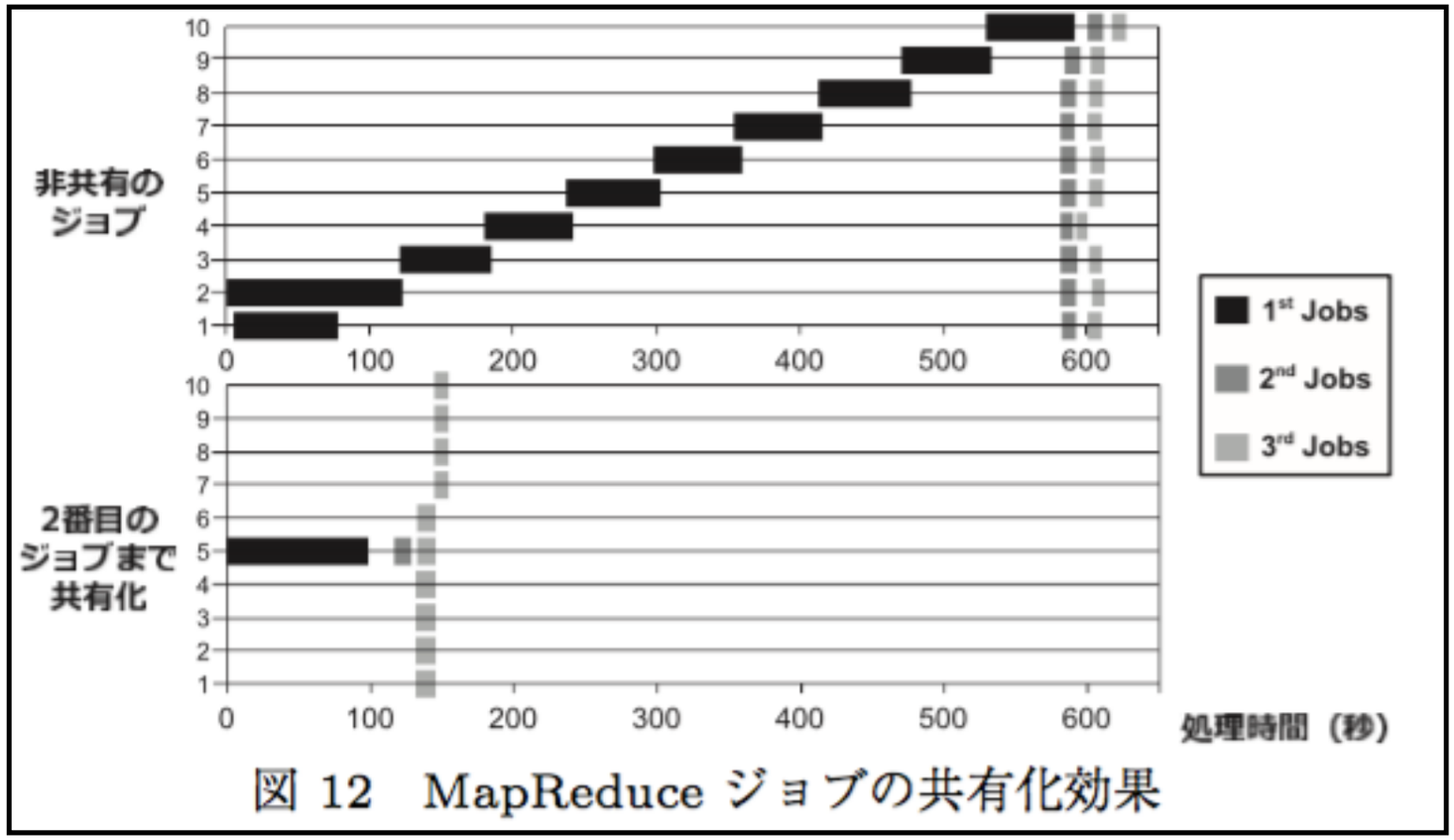


図 12　MapReduce ジョブの共有化効果

- 非共有に比べ約四倍の時間短縮
- ハイパラメータが末尾の処理列の末尾で使われれる場合、高い効果

反復タイプアルゴリズムの共有化

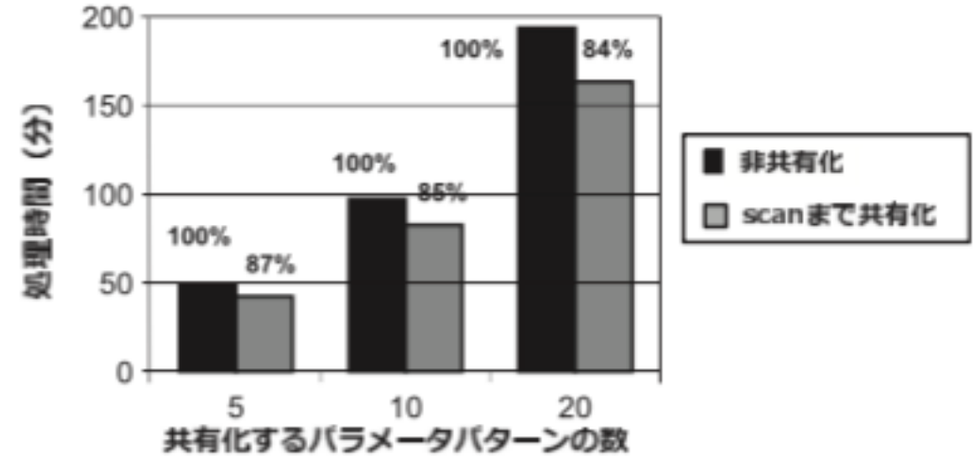


図 13　K-Means の scan 処理の共有化効果

- 20台のサーバクラスタ、反復回数は三回
- 約15%の高速化

むすび

- 本研究ではパラメータの影響部分を検知することで、自動的に複数の処理を共有可能部にマージできるフレームワークを提案した
- 総処理時間単出の効果が確認できた
- 説明変数（列）が一部重なりのある複数の機械学習処理を共有化することが今後の課題