Vol.2015-OS-135 No.2
Vol.2015-EMB-39 No.2
2015/11/24

IaaS環境におけるネットワーク帯域を考慮したMapタスクスケジューラ

尾板 弘崇[1,a)]　山田 浩史[1]

概要：大規模分散処理を行うフレームワークに MapReduce がある．MapReduce は分割された入力データに処理を施す Map タスクと，Map タスクの結果を収集したものに対して処理する Reduce タスクに分けられる．Reduce タスクは，自身が必要とする Map タスクの結果を各 VM からネットワーク経由で収集する Shuffle を行う．ソート処理など Map タスクの結果のデータ量が大きいときには，Shuffle に時間を要する．クラウド環境では物理マシンのリソースを複数の VM で共有しているために，Hadoop クラスタを構成する各 VM が利用可能なネットワーク帯域が異なる場合がある．各 VM の利用可能なネットワーク帯域が異なる環境で Shuffle 量が多い処理を行うと，利用可能なネットワーク帯域が狭い VM の Shuffle に時間がかかってしまい，ネットワーク帯域の広い VM は待機を余儀なくされてしまう．その結果，性能低下を引き起こしてしまう．この性能低下は，従来の MapReduce が各 VM のネットワーク帯域に関係なく，均一に Map タスクを割り当てるために起きる．本研究では，ネットワーク帯域に応じた MapReduce の Map タスクスケジューラを提案する．提案手法は，ネットワークに応じた MapOutput 量を各 VM に分配する．これによって，各 VM の Shuffle にかかる時間のズレを短縮し，性能が向上する．MapOutput を調節するために，提案手法は，ネットワーク帯域に応じて Map タスク数を調節する．Map タスク分配機構が動的に Slot を変更することで Map タスクのスケジューリングを調整し，Map タスク再実行機構が，ネットワーク帯域が動的に低下したときに Map タスクを他の VM で再実行することですでに生成した MapOutput 量を調節する．実験の結果，Shuffle-Heavy なワークロードで最大 25%の性能向上を実現した．

1. はじめに

クラウドコンピューティングは，ネットワーク経由で利用可能な大規模分散処理環境であり，マシンを自身で用意しなくても機械学習などの大規模な処理を行うことができる．一般にクラウド環境は仮想マシン (VM) を稼働させて運用することが多く，これにより，一台の物理マシンのリソースを複数の VM で共有している．実際に，Amazon EC2[1] は Instance と呼ばれる VM をユーザに貸し出しており，Amazon Elastic MapReduce[2] では，機械学習などに用いられる MapReduce[3] が展開されている Instance を提供している．

MapReduce は分散並列処理を行うフレームワークである．MapReduce は分割された入力データに対して行う Map タスクと Map タスクの結果を収集したものに対して行う Reduce タスクに処理を分割する．それぞれのタスクは各 VM に分散され，独立して処理を行うことによって並列分散処理を実現する．Reduce タスクは処理を行う前に，自身が必要とする Input データを収集する Shufffle を行う．Shuffle はクラスタを構成する全ての VM と通信を行い，Map タスクが生成した中間データの中から必要なデータを取得する．

MapReduce で実行される処理には，中間データの量が大きく，Shuffle に時間を要するものがある．そのような処理にはソートや候補作成などがある．例えば，検索エンジンでは，クローラーが Web 上から収集してきた全ての Web ページ上の全ての単語に対してソートを行い，巨大なデータベースに格納している．また，データ間の相関関係を発見するアソシエーション分析では膨大なデータの中からデータの組み合わせを作成するために候補作成が用いられる．

クラスタを構成する VM のネットワーク帯域が異なるとき，Shuffle にかかる時間が長くなり，性能低下を引き起こしてしまう．クラウド環境では，ネットワーク帯域を保証していない場合が多い．そのため，物理マシン上の VM の起動数などによって動的に各 VM が利用できるネットワーク帯域が変動してしまう [4][5][6]．そのような環境下

[1] 東京農工大学
Tokyo University of Agricluture and Technology
[a)] oita@asg.cs.tuat.ac.jp

Vol.2015-OS-135 No.2
Vol.2015-EMB-39 No.2
2015/11/24

で MapReduce で Shuffle 量の多い処理を実行した場合，実行時間が長くなってしまう．MapReduce はクラスタを構成するマシンが全て同一の性能であることを前提に設計されているため，Map タスクを均等に割り当て，各 VM に同量の MapOutput を生成する．しかし，それではネットワーク帯域の狭い VM 上の MapOutput の Shuffle に時間を要してしまう．各 VM 上の Reduce タスクはこの Shuffle の終了を待機させられるため，性能低下を引き起こす．

本研究では，ネットワーク帯域を考慮した MapReduce の Map タスクスケジューラを提案する．提案手法は各 VM が担う Shuffle データ量が VM のネットワーク帯域の比率になるように各 VM の MapOutput の生成量を調節する．そこで，各 VM に割り当てる Map タスク数を調節する．Map タスク提案手法は 2 つのキーデザイン (Map タスク分配機構，Map タスク再実行機構) をもつ．Map タスク分配機構は，Map Slot を動的に変更し，各 VM の Map タスクの並列数を調節することで，MapOutput の生成量を調節する．Map タスク再配置機構は，ネットワーク帯域が低下した場合，すでに終了した Map タスクを他の VM で再実行することでネットワーク帯域の低下した VM の Shuffle の負荷を低減する．

6 ノードのクラスタ上で Purdue MapReduce Benchmark Suite (PUMA)[7] を実行させることで本手法の効果を確認した．実験の結果，従来の Hadoop と比較して Shuffle-Heavy なワークロードでは最大 25%の性能向上が確認できた．また，本手法の効果が得られないと考えられる Shuffle-Light なワークロードでも 20%の性能低下に抑える事ができた．

本論文の残りの章は次のように構成される．2 章では，MapReduce の背景を紹介し，MapReduce の挙動や IaaS 環境での MapReduce の問題点について詳しく説明する．3 章では関連研究について述べる．4 章で，2 章で述べた問題点を解決するために提案する手法を述べ，5 章で提案手法のキーアイデアを詳しく説明する．6 章で実験を行い，提案手法の効果を確認する．そして最後に 7 章で本論文のまとめを述べる．

2. 背景と動機

2.1 MapReduce Mechanism

MapReduce は，並列分散処理フレームワークであり，大容量のデータを扱う場合に処理の高速化を実現する．機械学習やログ解析など大容量のデータを扱う場合，ディスク I/O が大きなボトルネックとなる．そこで，複数のマシンに処理を分散することで一つのマシンにかかるディスク I/O 量を減らし，それぞれのマシンが並列して処理を行うことで高速化が可能となる．

MapReduce のプログラミングモデルは Map 関数と Reduce 関数のみである．Map 関数は，Input データを key と value の組で構成された中間データに変換する．一方，Reduce 関数は，中間データを関連した key を持つ value をマージし任意の処理を行う．MapReduce を用いることで，並列分散処理に関する処理を自動的に行なってくれるため，容易に並列分散処理を行うことが可能となる．

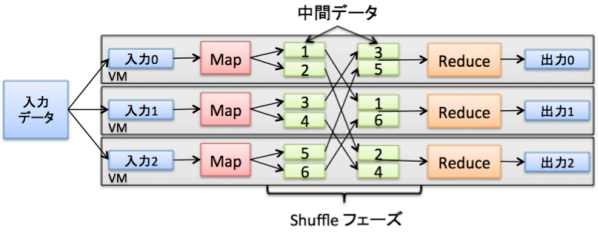

図 1　MapReduce のデータフロー

MapReduce は，入力データ，MapReduce のプログラム，設定情報から構成されるジョブを複数のタスクに分割し，実行することで並列分散処理を実現する．タスクは Map タスクと Reduce タスクの 2 種類で構成される．図 1 に MapReduce のデータフローを示す．MapReduce の実行モデルは 3 つのフェーズ (Map フェーズ，Shuffle フェーズ，Reduce フェーズ) で構成される．最初の Map フェーズは，自身に割り当てられた入力データ (MapInput) に対して定義した Map 関数を行う事によって，MapInput を複数の<key，value>に変更し，中間データ (MapOutput) を生成する．Hadoop では Combiner 関数も実装することができる．Combiner 関数が実装された場合，Map フェーズにおいてそれぞれの Map タスクが生成した MapOutput 内で集計を行うことで MapOutput の容量を削減することができるようになる．2 番目の Shuffle フェーズは，Reduce タスクが割り当てられた key を持つ MapOutput を収集し，ReduceInput を作成する．一般に生成される key の数は Reduce タスクの総数よりも多いため，1 つの Reduce タスクが複数の key を割り当てられる．最後の Reduce フェーズでは，key ごとにまとめられた value に対して Reduce 関数に定義した処理を行い，最終的な結果を求める．

Map フェーズと Reduce フェーズは Hadoop クラスタ上で並列して行われる．Map フェーズは Map タスクによって行われる．一方で，Shuffle フェーズから Reduce フェーズまでは Reduce タスクで行われる．Reduce タスクは MapOutput を保持する VM にデータを送信するよう依頼することによって MapOutput を取得する．Map タスクは Input データの内容を考慮せずに VM に割り当てられるため，特定の key を持つ MapOutput は様々な VM に分散されることになる．そのため，一般的に Shuffle は全ての VM が全ての VM と通信を行う．また，Reduce タスクは自身が担当する MapOutput の Shuffle が全て終了するまで Reduce フェーズに移行することができない．

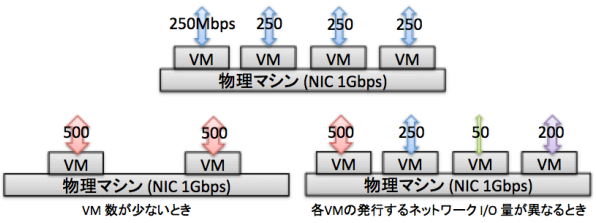

図 2　VM 数による利用可能帯域の変化

2.2　動的なネットワーク帯域の変化

MapReduce はクラウド環境で広く使われている．実際，Amazon Web Service[1] では Amazon Elastic MapReduce[2] という MapReduce 専用のインスタンスを提供するサービスを行っている．クラウド環境では多くの場合，仮想マシン (VM) を用いている．一台の物理マシンのリソースを複数の VM で共有しているため，物理マシン上の VM の数や挙動によってそれぞれの VM が利用可能なリソース量が動的に変化する．例えば，図 2 のように，1Gbps の物理 NIC がある物理マシン上で 4 台の VM が稼働している場合，各 VM は　250Mbps のネットワーク帯域を利用できる．一方で，同様の物理マシン上で 2 台の VM が稼働している場合，各 VM は 500Mbps のネットワーク帯域を利用することができる．そしてクラウド環境では VM の起動や終了が行われるたびに VM が利用可能なネットワーク帯域が変化する．また，物理マシン上の VM 数が一定でもアプリケーションの動作によっても VM それぞれが利用可能なネットワーク量が動的に変化する．例えば，図 2 のように，同一の物理マシン上で稼働している 4 台の VM のうち，3 台の VM がそれぞれ利用しているネットワーク帯域が 250Mbps，50Mbps，200Mbps の場合，残りの VM は 500Mbps のネットワーク帯域を利用することができる．よって，VM 上で実行しているアプリケーションの変化やアプリケーションの使用リソースの変化等によって VM が利用可能なリソース量が変化する．

2.3　仮想クラスタにおける MapReduce

MapReduce クラスタを構成する VM のネットワーク帯域が異なる場合，従来の MapReduce では，Reduce タスクが各 VM から MapOutput を取得する時間にばらつきが生じてしまう．その結果，Reduce タスクは利用可能帯域の狭い VM からの MapOutput の取得が終了するまで待機を余儀なくされる．結果，同じ総ネットワーク帯域量を各 VM に均等に与えられた場合と比べて Shuffle フェーズにかかる時間や与えられた CPU などのリソースを利用できない時間の長期化といった悪影響を生じさせてしまう．

各 VM の利用可能帯域の異なる環境における Reduce タスクの MapOutput の取得にかかる時間のばらつきは，各 VM の Shuffle データ量 (各 VM が生成した MapOutput の総量) が各 VM のネットワーク帯域と異なる比率で生成さ

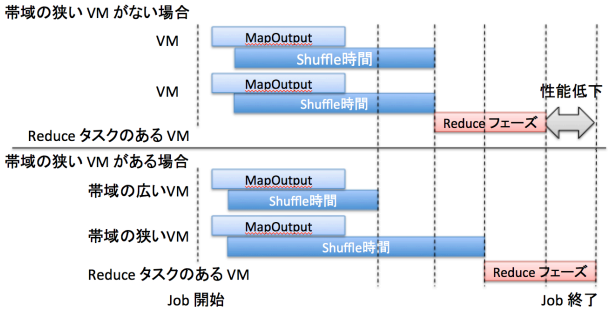

図 3　各 VM のネットワーク帯域の等しい場合と異なる場合の MapReduce

れるために起こる．Hadoop は Slot が空いたところからタスクを実行する．Map タスクはタスク内で処理を完結するため，各 VM で行われる Map タスク数は各 VM のネットワーク帯域を考慮しない．ノンローカルな Map タスクの場合はネットワークも関係してくるが，Fraz らによって同じ性能のマシンで構成されたクラスタで実行した場合，95%の Map タスクがローカリティを満たすことを明らかにしている．よって，ネットワーク帯域が異なる場合においても，各 VM の帯域幅に関係なく，ほぼ同等の数の Map タスクが実行され，MapOutput が生成される．しかし，この場合，帯域の広い VM に比べて帯域の狭い VM では Shuffle に時間がかかってしまう．Reduce タスクは Shuffle が全て終了するまで Reduce フェーズに移行することができないため，Reduce タスクは帯域の狭い VM からの MapOutput の Shuffle を待機し続けなければならず，その間，Reduce タスクが割り当てられている VM の Slot を専有し続ける．よって実行時間が長くなってしまうばかりか，他のタスクを実行できなくなってしまう．例えば，図 3 のように，ネットワーク帯域の狭い VM がない場合は，それぞれの VM に同量の MapOutput が分配したとき，ほぼ同時に Shuffle フェーズが終了し，Reduce フェーズが開始されるが，ネットワーク帯域の狭い VM がある場合，帯域の狭い VM 上の MapOutput の Shuffle に時間がかかり，Reduce フェーズはその Shuffle の終了を待機させられる．ネットワーク帯域の広い VM 上の Reduce タスクの Shuffle フェーズにかかる時間がネットワーク帯域の狭い VM 上の MapOutput の Shuffle によって長くなってしまう．

3.　関連研究

ヘテロジニアスな Hadoop クラスタにおける最適化を行った研究がある．Tarazu[8] は，ノンローカルタスクの MapInput の取得によるネットワーク負荷を軽減した研究である．ノンローカルな Map タスクの実行の抑制や分散して実行させるなどのスケジューリングを行うことで，ノンローカルタスクによるネットワーク負荷を軽減し，Shuffle フェーズの長期化を抑制している．本研究が想定している環境は，ノンローカルタスクが大量に発生する環境では

ないため，ノンローカルタスクのスケジューリングを行う Tarazu では本論文で掲げる問題を解決することができない．LATE[9] は，ヘテロジニアス環境で発生しやすい Hadoop の投機的実行によって生じるリソースの無駄遣いや性能低下を改善した研究である．Map タスクの進行状況の分析し，投機的実行タスクのスケジューリングを行うことで不必要な投機的実行によって性能が低下するのを防止している．残念ながら，MapOutput の Shuffle は Map タスクの終了後に行われるため，Map タスクの投機的実行は適応されず，Reduce タスクの投機的実行を行っても MapOutput の送信元は変わらないため，各 VM で利用可能帯域が異なる場合の非効率さを解消することはできない．ANT[10] は，ヘテロジニアスクラスタにおいて，それぞれのタスクの最適化を行った研究である．それぞれのタスクに対して異なる Configuration の設定を実現し，ヘテロジニアスクラスタを構成するそれぞれのマシンに最適な Configuration で実行できるようにすることでそれぞれのタスクの高速化を実現した．この研究はタスクの高速化が目的であるため，ヘテロジニアスクラスタでは Shuffle に時間がかかってしまうという本論文の問題を解決することはできない．

本研究では，クラウド環境における競合によって生じる問題に対処しているが，競合に対処する方法を提案した先行研究が幾つか存在する．Q-Cloud[11] は，VM に与えるリソースの動的な調整を提案した研究である．達成すべき性能目標と現在の性能とを考慮してそれぞれの VM に CPU を動的に割り当てることで，競合による性能低下に対処している．TRACON[12] や ILA Scheduler[13] は，競合の影響が最小限になるようにタスクをスケジューリングする手法を提案した研究である．Map タスクを割り当てる際，各 VM の現在のリソース使用量やタスクの特徴などを分析し，競合の影響が小さいタスクを割り当てることで，性能低下を防いでいる．残念ながら，競合への対処法を提案した研究は，CPU やディスク I/O の競合を対象としており，ネットワーク I/O の競合に対処した研究は多くない．そのため，ネットワーク I/O が競合する環境において本手法は高い効果を発揮すると考えられる．

本研究の目標は Shuffle フェーズの高速化である．MapReduce の Shuffle フェーズの高速化に対処した研究は，他にもいくつかある．ShuffleWatcher[14] は，複数のジョブが同時に実行されている際に，それぞれのジョブを MapReduce クラスタ内のネットワーク距離の近いマシンにまとめて割り当てることで Shuffle フェーズの高速化を実現している．一方，Purlieus[15] は，MapReduce クラスタを起動する際に VM の配置をネットワーク距離の近い物理マシンにまとめることで Shuffle フェーズの高速化を実現している．iShuffle[16] は Reduce タスクから Shuffle フェーズを分離することで Shuffle フェーズの間，Reduce タスクが無駄にリソースを専有してしまうことの解消と Map フェーズと Shuffle フェーズのオーバラップの性能向上を実現した．これらの研究は，Shuffle の高速化を実現したが，本研究で掲げるネットワーク帯域が動的に変動することには対処できない．

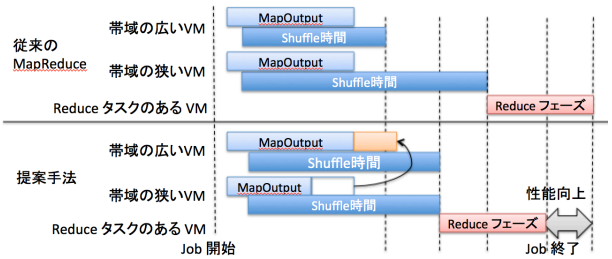

図 4　Shuffle 時間

他にもタスクの高速化の観点から MapReduce の性能向上を実現した研究が多くある．例えば，データローカリティに着目して性能向上を実現した研究がある [13], [17]．これらの研究は複数のジョブが同一のクラスタ内で動作してる場合，公平性を犠牲に，input があるノードに Map タスクを割り当てることで input を取得するためのタイムロスを低減する．他にも，ANT と同様に Configuration の最適化を行うことで性能向上を実現した研究がある [18], [19]．これらの研究はタスクの高速化に着目しているため，ヘテロジニアスクラスタにおいて各 VM の MapOutput の Shuffle にかかる時間に差が生じるのを解決することはできない．しかし，一部制限をかける必要があるが，本手法とこれらの手法を同時に用いることでさらなる性能向上を実現できると考えられる．

4. 提案

本論文はネットワーク帯域を考慮した Map タスクスケジューラを提案する．今まで述べてきたように，各 VM のネットワーク帯域の異なる環境において，従来の Hadoop では，各 VM のネットワーク帯域に関係なく，Map タスクを割り当てられ，MapOutput が生成させるため，各 VM にある MapOutput を取得し終わるまでの時間にばらつきが生じてしまい，結果として非効率な実行になってしまっていた．そこで，提案手法は，図 4 のように，各 VM が担う Shuffle データ量が各 VM のネットワーク帯域の比率になるようにする．これによって，各 VM の Shuffle フェーズにかかる時間の違いが短縮され，ネットワーク帯域の広い VM が狭い VM の Shuffle フェーズを待機することを防ぐことで，非効率なリソース利用を軽減することが可能になる．

提案手法は，図 5 のように，各 VM に各 VM のネットワーク帯域の比率を考慮して Map タスクを割り当てることで各 VM の Shuffle データ量を調節する．ネットワーク帯域に応じて Slot を動的に変更することで，Map タスク

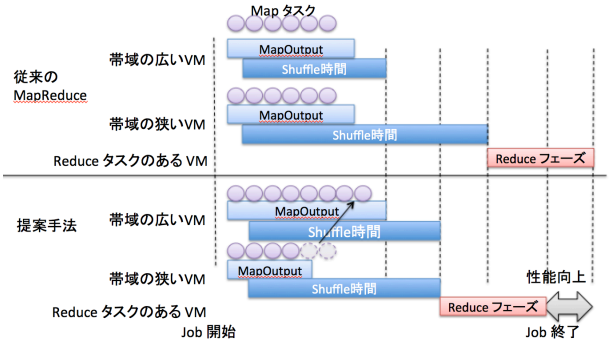

図 5　提案手法の概念

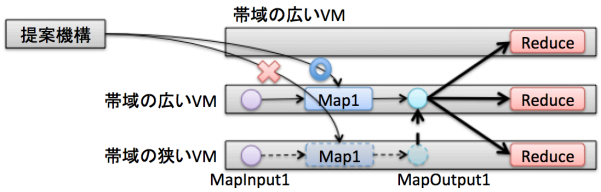

図 6　MapOutput の流れ

の実行ペースを遅くする，あるいは，一定数以上の Map タスクの実行を妨げることでネットワーク帯域に応じた Map タスクのスケジューリングを行う．また，ネットワーク帯域が著しく低下した場合，実行しすぎた Map タスクを他の VM で再実行することでネットワーク帯域に応じた MapOutput 量に調節する．

MapOutput を移動することで各 VM の Shuffle データ量を調節するのではなく，Map タスクの割り当て方によって調節することで不必要なネットワーク I/O が生じるのを抑えることができる．図 6 は，MapOutput を移動させる場合と Map タスクの割り当てを調節する場合の MapOutput の流れを示している．MapOutput を移動させる場合，移動元の VM は MapOutput の移動のために移動しない場合と同量のネットワーク I/O を発生させる上に，移送先の VM も Shuffle フェーズによるネットワーク I/O を発生させるため，2 重にネットワーク I/O を発生させてしまう．一方で，MapOutput を移動せず，Map タスクの割り当て先を調整する場合，MapOutput の移動によるネットワーク I/O は発生しない．さらに，HDFS の複製のある VM に Map タスクを割り当てることで MapInput の取得によるネットワーク I/O も発生しない．よって，Map タスクの割り当て先を調節する場合，発生するネットワーク I/O は Shuffle によるネットワーク I/O のみとなる．

Map タスクのスケジューリングを行うことによって MapOutput の送信側の送信量はネットワーク帯域と比例するようになるが，受信側の受信量はネットワーク帯域に比例しない．受信側の受信量を調節するためには，Reduce タスクのスケジューリングを行う必要があるが，本研究では Reduce タスクのスケジューリングは行わない．Reduce タスクは Map フェーズとのオーバーラップのために全ての Map タスクが終了する前にスケジューリングされる．Reduce タスクをネットワーク帯域に応じてスケジューリングするためには，各 Reduce タスクの Input サイズが必要となるが，MapOutput が作成される前に MapOutput が各 Reduce タスクにどのように分配されるのかを推定するのは困難である．よって，ネットワーク帯域に応じた Reduce タスクのスケジューリングは困難である．また，ネットワーク帯域が異なる環境でネットワーク帯域に応じて MapOutput を割り当てた場合，ネットワーク帯域の狭い VM に割り当てられた Reduce タスクは MapOutput を均等に割り当てられた場合に比べてネットワークを通して取得する MapOutput 量が増えるため，Shuffle フェーズに時間を要してしまう．しかし，この場合，Hadoop は投機的実行を行われ，自動的に他の VM で再実行される．結果，Reduce タスクはネットワーク帯域の広い VM に割り当てられるため，その Reduce タスクもネットワーク帯域に応じた MapOutput の割り当ての恩恵を受けられ，高速化すると考えられる．これらの理由から Reduce タスクのスケジューリングは行わない．

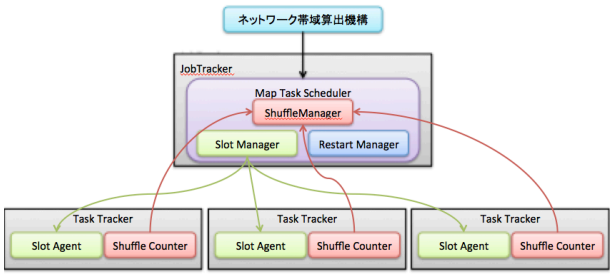

図 7　提案手法の全体図

5. デザイン

これらを実現するために，提案手法は，次のように，3 つの要素から構成される．図 7 に提案手法の全体像を示す．

- ネットワーク帯域算出機構
  各 TaskTracker の利用可能ネットワーク帯域を求める．
- Map タスク分配機構 (Slot Manager，Slot Agent)
  各 TaskTracker の Slot の値を動的に変更する
- Map タスク再実行機構 (Restart Manager，Shuffle Manager，Shuffle Counter)
  再実行する Map タスクを決定する

5.1 ネットワーク帯域算出機構

ネットワーク監視機構は，VMM から物理マシン上の VM のネットワーク情報を収集する．提案手法はネットワーク帯域に応じて Map タスクの割り当てを行う．仮想化環境を利用している場合，各 VM はネットワーク帯域を共有しているため，物理マシンの最大ネットワーク帯域をそのまま利用できるわけではない．よって提案手法は，物

理マシン上の各 VM のネットワーク情報を集めることで，物理マシンのネットワーク帯域幅のうち，特定の VM が利用できると思われる帯域幅 (利用可能帯域幅) を見積もる．チャレンジングな点は，VM は同じ物理マシン上の VM の情報へのアクセスを行わないようにする点である．提案手法は新たなネットワーク情報集約サーバと VMM で構成されるネットワーク監視機構を用意することで，機密性を保持している．

5.2 Map タスク分配機構

Hadoop は起動時に各 TaskTracker の Slot を一意の値に設定する．ある VM に利用可能な MapSlot が生じたとき，JobTracker はその VM に Map タスクを割り当てていく．Map タスクは同じデータ量の入力データに同様の処理を行う．そのため，同様のペースで同様の MapOutput を作成する．そのため，各 VM は，利用可能帯域にかかわらず，同様の Shuffle データ量を生成してしまう．そこで，提案手法は，各 VM の利用可能帯域に応じたペースで Shuffle データを生成するよう，Map タスクのスケジューリングを行う．提案手法は，各 TaskTracker に設定されている Slot の最大値を TaskTracker ごとに異なる値を割り当てることで，Map タスクを実行するペースを調整する．これによって，利用可能帯域の狭い VM では，広い VM と比較して Shuffle データ量を少なくすることができる．

Map タスク分配機構は，図 7 のように JobTracker 内の Slot Manager と TaskTracker 内の Slot Agent によって構成される．Slot Manager は，JobTracker がネットワーク監視機構から取得したクラスタのネットワーク情報や Map タスクの実行状況を元に Slot 数を決定する．その後，決定した Slot 数を Heartbeat に追加することで Slot Agent に通知する．Slot Agent は通知された値で TaskTracker の Slot 数を更新する．

利用可能帯域を元にした Slot 数の決定方法を Algorithm1 に示す．Slot 数は起動時に設定した Slot 数に対して対象の VM の利用可能帯域とクラスタ内の VM で最大の利用可能帯域の比率を掛けたもので算出する．Slot 数は Map Slot と Reduce Slot を合わせて，そのマシンが利用可能な CPU の数にするのが一般的である．Slot 数を CPU 数以上にすれば並列実行数は増加するが，タスクを実行している JVM は CPU を共同で利用することになり，必ずしも性能が向上しない．そのため，予め CPU 数を元に設定されている初期値を基準に設定することで．CPU の処理能力以上のタスクを割り当てることを防ぐ．

5.3 Map タスク再実行機構

一度実行した Map タスクを改めて再実行する．クラウド環境では，ネットワーク帯域は動的に変動する．一部の VM の利用可能帯域が減少した場合に，これまで作成された MapOutput は減少する前のネットワーク帯域を元に作成量が決定されているため，減少したネットワーク帯域で Shuffle すると同等の MapOutput を保持する VM に比べて時間がかかってしまう．そこで，Shuffle が行われている場合でも一度終了した Map タスクを他の VM で再実行することで，それぞれの VM が同時に Shuffle を終了するように Shuffle 量を再調整する．

Algorithm 1 利用可能帯域の決定方法

```
MAX : MaxNetworkBandWidthofPhysicalNIC
Average : Max/numberofVMs
Current : currentVMNetworkBandWidth
Available : AvailableNetworkBandwidth
if Current <= Average then
  Available ← Average
else
  Available ← Current + (Max − VMsCurrent)/VMs
end if
```

Map タスク再実行機構は，Shuffle された回数の少ない MapOutput を作成した Map タスクを再実行する．そのため，各 MapOutput の Shuffle 回数を計測する． Shuffle Counter は<MapTaskID，Shuffle Count>の組を保持している．Tracker は Reduce タスクからリクエストを受け，MapOutput を送信したとき，Shuffle Counter を呼び出し，その MapOutput と対応する MapTaskID の value に 1 を加算することで Shuffle 回数を計測する．TaskTracker は Heartbeat を用いて Shuffle Counter が持つ Shuffle 情報を JobTracker に送信する．JobTracker は TaskTracker ごとに受け取った Shuffle 情報を保存する．JobTracker は，クラスタを構成する VM の利用可能帯域の減少を感知したとき，Restart Manager を起動し，その VM の Shuffle 情報を Restart Manager に受け渡す．Restart Manager は Shuffle 情報を用いて Shuffle 回数の少ない Map タスク抜き出し，ネットワーク帯域の減少量を元に再実行する Map タスクの数を決定する．そして，JobTracker の Task Scheduler にそのタスク情報を渡し，再実行を依頼する．

6. 実験

6.1 実験環境

実験では，6 ノードの仮想クラスタを用いる．3.3GHz，6 コアの Intel Xeon プロセッサ E3-1230，32GB のメモリを搭載した物理マシン一台につき 1 つの VM が稼働している．それぞれの VM には 6vCPUs と 16GB のメモリを割り当てている．それぞれの物理マシンは 1Gbps のイーサネットで通信を行う．Linux カーネルは 3.16.0 で VMM は Xen 4.4 を用いた．

提案手法は Hadoop 1.2.1 上に実装を行った．1 つの VM 上で Master と JobTracker が，残りの 5 つの VM 上で DataNode と TaskTracker が動作している．Hadoop の設定はハードウェアの設定を元に，4 Map Slot と 2 Reduce

Slot，他の設定は全てデフォルトとした．ネットワーク帯域算出機構は Mastar が動いている VM のある物理マシン上で稼働している．

提案手法を評価するにあたって Purdue MapReduce Benchmark Suite(PUMA)[7] を用いた．PUMA は様々な MapReduce のベンチマークと実際に使われているようなデータセットを提供している．表 1 に用いたベンチマークとその詳細を示している．

これらのベンチマークは 3 つのカテゴリ (Shuffle-Heavy，Shuffle-Medium，Shuffle-Light) に分類した．これらは入力データに対する Shuffle データ量によって分類している．

6.2 静的環境下での性能評価

一部の VM のネットワーク帯域が狭く，それぞれのネットワーク帯域が動的に変動しない場合の提案手法の効果を確認する．Linux に搭載されている機能を用いて，3 つの VM のネットワーク帯域を 200Mbps に 2 つの VM のネットワーク帯域を 50Mbps に制限することで静的環境を実現した．図 8 に Tera-Sort の各 VM で生成された Shuffle データ量を示している．図 8 より，帯域の狭い VM の Shuffle データ量を削減できていることが確認できる．ネットワーク帯域の広い VM と狭い VM のネットワーク帯域の比率と実際の Shuffle データ量の比率が異なってしまっているが，これは，Map タスクを並列実行した際に Disk I/O の競合により，帯域の狭い VM の Map タスクが広い VM と比較して高速化したためである．そのため，Disk I/O の競合を抑えることによって，より正確な分配が可能となる．

図 9 に変更を加えていない Hadoop での実行時間で正規化した提案手法の実行時間を示している．Shuffle-Heavy なワークロードは全て高速化した．しかしながら，Tera-Sort と Self-Join は 25%高速化したのに対し，Ranked-Inverted-Index は 5%の高速化にとどまった．図 10，図 11 はそれぞれ Tera-Sort と Ranked-Inverted-Index における変更を加えていない Hadoop(下) と提案手法 (上) の各 Reduce タスクのフェーズの推移を示している．これより，Tera-Sort は帯域の広い VM に割り当てられた Reduce タスクが終了した後に帯域の狭い VM に割り当てられた Reduce タスクが投機的実行によって帯域の広い VM に割り当てられたことで全ての Reduce タスクが高速化している．一方で，Ranked-Inverted-Index は帯域の広い VM に割り当てられた Reduce タスクの Shuffle は高速化しているが，Reduce フェーズが長いため，Reduce Slot が空かず，投機的実行が行えていない．これにより，Ranked-Inverted-Index の高速化が Tera-Sort や Self-Join よりも小さくなったと考えられる．

Shuffle-Medium なワークロードは約 10%性能が低下した．図 12 に Inverted-Index の変更を加えていない Hadoop と提案手法の各 Reduce タスクのフェーズの推移を示して

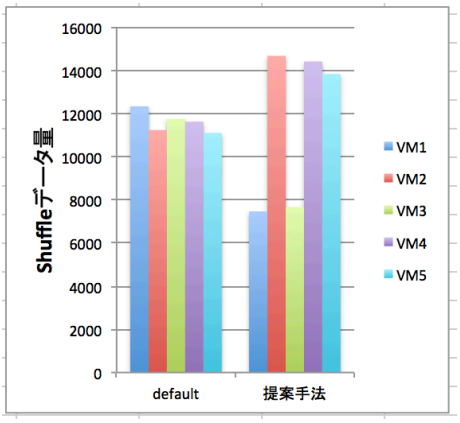

図 8　各 VM の Shuffle データ量

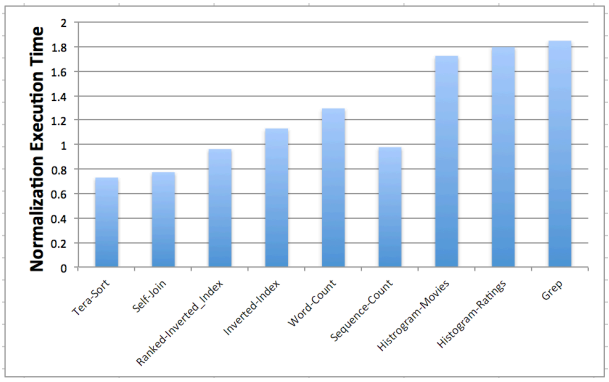

図 9　ネットワーク帯域が動的に変動しない環境下における実行時間

いる．図 12 より，提案手法の方が Shuffle に時間がかかってしまっている．Inverted-Index では，Shuffle データの量が小さいため，帯域の広さに関係なく，同じ速度で Shuffle できてしまう．そのため，提案手法は Map タスクが終了し，新たな MapOutput が生成されるのを待機しなければならなくなってしまう．

Shuffle-Light なワークロードは変更の加えていない Hadoop と比較して約 20%性能が低下した．性能低下の要因として 2 つが挙げられる．1 つ目が，提案手法は Slot を減らすため，Map タスクの並列数が減少し，Map フェーズの時間が長くなってしまう．Shuffle-Light なワークロードは Shuffle データ量が少なく，実行時間の大部分を Map タスクの実行が占めるため，その影響を大きく受けてしまう．2 つ目は提案手法のオーバヘッドである．提案手法では定期的にネットワーク帯域算出機構にネットワークを通じて通信を行うため，一定のオーバヘッドが生じてしまう．

6.3 動的環境下での性能評価

動的に一部の VM のネットワークが変動する環境下での提案手法の効果を確認する．5VM のうち 2VM は 200Mbps で固定する．残りの 3VM のうち，2VM を 50Mbps に設定し，100 秒ごとに 50Mbps に設定する VM を入れ替えていくことで動的環境を構成した．図 13 に Tera-Sort の各 VM

表 1　PUMA ベンチマーク

Benchmark	Input Size(GB)	Input data	Maps	Reduces	Shuffle Volume(GB)	Type
Tera-Sort	60	synthetic	894	10	61	Shuffle-Heavy
Self-Join	30	synthetic	448	10	29	Shuffle-Heavy
Ranked-Inverted-Index	40	Sequence-Count	768	10	45	Shuffle-Heavy
Inverted-Index	50	Wikipedia	768	10	10	Shuffle-Medium
Word-Count	50	Wikipedia	768	10	10	Shuffle-Medium
Sequence-Count	50	Wikipedia	768	10	10	Shuffle-Medium
Histogram-Movies	30	netflix data	428	10	0.00006	Shuffle-Light
Histogram-Rating	30	netflix data	428	10	0.00004	Shuffle-Light
Grep	50	Wikipedia	788	10	0.00005	Shuffle-Light

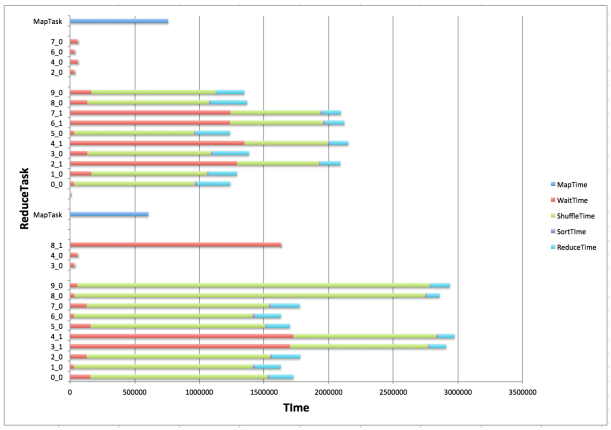

図 10　Reduce タスクのフェーズの推移 (Tera-Sort)

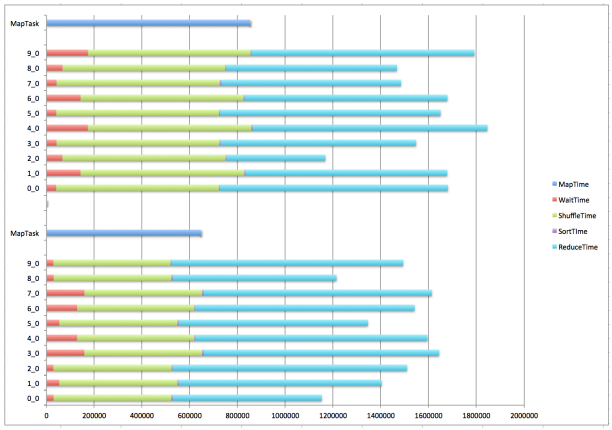

図 12　Reduce タスクのフェーズの推移 (Inverted-Index)

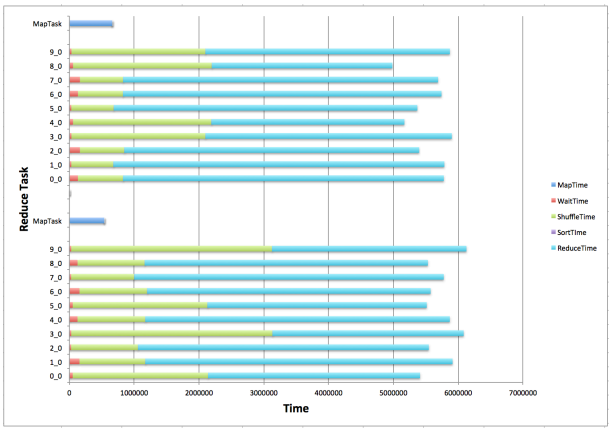

図 11　Reduce タスクのフェーズの推移 (Ranked-Inverted-Index)

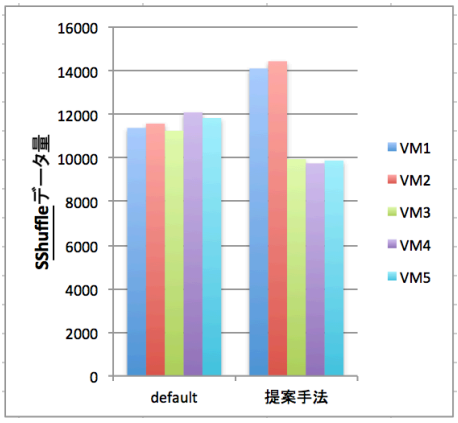

図 13　各 VM の Shuffle データ量

で生成された Shuffle データ量を示している．図 13 より，帯域の狭い VM の Shuffle データ量を削減できていることが確認できる．

図 14 に変更を加えていない Hadoop での実行時間で正規化した提案手法の実行時間を示している．Shuffle-Heavy なワークロードは Tera-Sort と Self-Join は約 15%，Ranked-inverted-Index は 5%高速化した．一方で，Shuffle-Medium なワークロードは平均で 5%，Shuffle-Light なワークロードは 20%の性能低下した．これらの原因は静的環境下における性能低下と同様であると考えられる．

7. おわりに

MapReduce はホモジニアス環境を想定して設計されているため，クラウド環境のようなネットワーク帯域が動的に変動する環境では最適な性能を発揮することができない，そこで本研究では，ネットワーク帯域が動的に変動する環境で Shuffle フェーズを高速化する手法を提案した．提案手法はネットワーク帯域に応じた Map タスクのスケジューリングを行う．Hadoop 上に実装し，6 ノードクラスタで評価を行った．実験の結果，Shuffle-Heavy なワークロードで最大 25%の性能向上を実現した．

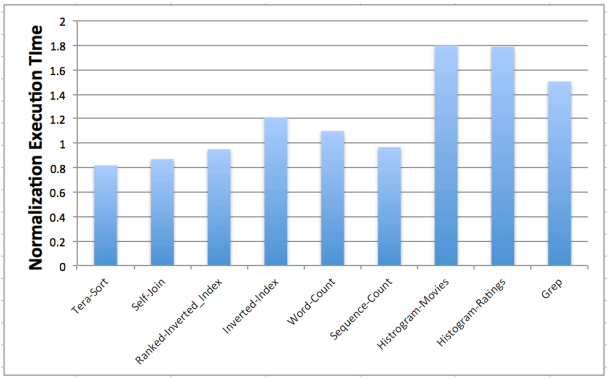

図 14　ネットワーク帯域が動的に変動する環境下における実行時間

参考文献

[1] Amazon: Amazon Web Service (AWS) - Cloud Computing Service. `http://aws.amazon.com/`.

[2] Amazon: AWS | Amazon Elastic MapReduce (EMR) | Hadoop MapReduce in the Cloud. `https://aws.amazon.com/jp/elasticmapreduce/`.

[3] Dean, J. and Ghemawat, S.: MapReduce: simplified data processing on large clusters, *Proc of the 6th Conference on Symposium on Operating Systems Design and Implementation (OSDI '04)*, pp. 10–10 (2004).

[4] Shea, R., Wang, F., Wang, H. and Liu, J.: A Deep Investigation Into Network Performance in Virtual Machine Based Cloud Envirtonments, *Proc. of the 33th IEEE International Conference on Computer Communications (INFOCOM '14)*, pp. 1285–1293 (2014).

[5] Wang, G. and Ng, T. S. E.: The Impact of Virtualization on Network Performance of Amazon EC2 Data Ceter, *Proc. of the 29th IEEE International Conference on Computer Communications (INFOCOM '10)*, pp. 1–9 (2010).

[6] Schad, J., Dittrich, J. and Quiane-Ruiz, J.-A.: Runtime measurements in the cloud: observing, analyzing, and reducing variance, *Proc of the VLDB Endowment(VLDB '10)*, pp. 460–471 (2010).

[7] Ahmad: pumabenchmarks. `https://engineering.purdue.edu/~puma/pumabenchmarks.htm`.

[8] Ahmad, F., Chakradhar, S., Raghunathan, A. and Viavkumar, T. N.: Tarazu: Optimizing MapRedece On Heterogeneous Clusters, *Proc of the 18th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '12)*, pp. 61–74 (2012).

[9] Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R. and Stoica, I.: Improving mapreduce performance in heterogeneous environments, *Proc. of the USENIX Symposium on Operating Systems Design and Implementation (OSDI '08)*, pp. 29–42 (2008).

[10] Cheng, D., Rao, J., Guo, Y. and Zhou, X.: Improving MapReduce Performance in Heterogeneous Environments with Adaptive Task Tuning, *Proc of the 15th International Middleware Conference (Middleware '14)*, pp. 97–108 (2014).

[11] Nathuji, R., Kansal, A. and Ghaffarkhah, A.: Q-Clouds:managing performance interference effects for QoS-aware clouds, *Proc of the 5th European Conference on Computer systems (EuroSys '10)*, pp. 237–250 (2010).

[12] Chiang, R. C. and Huang, H. H.: TRACON: Interference-Aware Scheduling for Data-Intensive Applications in Virtualized Environments, *Proc of the International Conference for High Performance Computing, Networking, Strage and Analytics (SC '11)*, pp. 1–12 (2011).

[13] Bu, X. and Rao, J.: Interference and locality-aware task scheduling for MapReduce applications in virtual clusters, *Proc of the 22nd International Symposium on High-performance Parallel and Distributed Computing (HPDC '13)*, pp. 227–238 (2013).

[14] Ahmad, F., Chakradhar, S. T., Raghunathan, A. and Vijaykumar, T. N.: ShuffleWatcher: Shuffle-aware scheduler in multi-tenant MaReduce clusters, *Proc of the 2014 USENIX Conference on USENIX Annual Technical Conference (USENIX ATC '14)*, pp. 1–12 (2014).

[15] Palanisamy, B., Singh, A., Liu, L. and Jain, B.: Purlieus: Locality-aware Resource Allocation for mapreduce, *Proc of the International Conference for High Performance Computing, Networking, Strage and Analytics (SC '11)*, pp. 1–11 (2011).

[16] Guo, Y., Rao, J. and Zhou, X.: iShuffle: Improving Hadoop Performance with Shuffle-on-Write, *Proc of the 10th USENIX/ACM International Conference on Autonomic Computing (ICAC '13)*, pp. 107–117 (2013).

[17] Zaharia, M., Borthakur, D., Sarma, J. S., Elmeleegy, K., Shenker, S. and Stoica, I.: Delay scheduling: a simple technique for achieving locality and fairness scheduling, *Proc of the 5th European Conference on Computer Systems (EuroSys '10)*, pp. 265–278 (2010).

[18] Lama, P. and Zhou, X.: AROMA: automated resource allocation and configuration of mapreduce environment in the cloud, *Proc of the 9th International Conference on Autonomic Computing (ICAC '12)*, pp. 63–72 (2012).

[19] Li, M., Zeng, L., Meng, S., Tan, J., Zhang, L., Butt, A. R. and Fuller, N.: MRONLINE: MapReduce online performance tuning, *Proc of the 23rd International Symposium on High-performance Parallel and Distributed Computing (HPDC '14)*, pp. 165–176 (2014).