Vol.2014-ARC-210 No.11
Vol.2014-OS-129 No.11
2014/5/15

MapReduceにおける通信データ量を考慮したReduceタスク割り当て手法

阪東 幸二[1]　芝　公仁[1]

概要：MapReduce では，データ処理を複数の Map タスクと Reduce タスクに分割し，各ノードに分散して並列に処理を行う．Reduce タスクは Map タスクによって出力されたデータを入力とするため，他のノードからネットワークを介したデータのコピーが必要となる．このときのコピーするデータ量を削減することで，効率的に処理することができる．Hadoop MapReduce における Reduce タスクの割り当ては，他のノードからコピーする通信データ量を考慮していない．本稿では，各ノードが格納する Map タスクの出力データ量を取得し，通信データ量を考慮した Reduce タスクの割り当てを行う手法について述べる．本手法によって，クラスタ全体での通信データ量を削減し，効率的にタスクを処理することが可能になる．

1. はじめに

近年，インターネットの普及とサービスの多様化により，生成されるデータは日々増大しており，それに伴い処理するデータも大規模化している．しかし，1 台の計算機では処理能力に限界があり，これらの大規模データを処理するには非常に多くの時間を要する．また，大規模データを高速に処理するための高性能な計算機は非常に高価である．そこで，安価な複数の計算機をネットワークで接続し，処理を分散して並列に動作させることで，処理能力を向上させる並列分散処理が注目されている．しかし，並列分散処理では，計算機間のデータ通信処理や障害に対する処理など，実際に行いたいデータ処理以外の処理が非常に多く繁雑である．このような並列分散処理をより容易に行うための環境として，Hadoop[4] が開発されている．

Hadoop は分散型コンピューティングのためのオープンソースソフトウェアである．Hadoop は並列分散処理フレームワーク MapReduce[5] と分散ファイルシステム Google File System[6] の概念をもとに開発されたもので，それぞれ Hadoop MapReduce と Hadoop Distributed File System として開発が行われている．Hadoop MapReduce は，分散処理に必要なノード間の通信処理や，タスクの割り当て，障害に対する処理を行う機能を提供しており，ユーザは，入力を並列に処理する Map と，その結果を集約する Reduce という 2 つの処理を記述するだけで並列分散処理を行うことができる．

[1] 龍谷大学
Ryukoku Univercity

しかし，Hadoop にはいくつかの問題点が存在している．その問題点の 1 つとして Reduce タスクの割り当てが効率的でないことが挙げられる．Reduce タスクの割り当ては，Map タスクの割り当てのようなデータローカリティを考慮した実装が行われておらず，ノード間で多くのデータの送受信が必要となるタスク割り当てが行われる場合がある．この結果，ネットワークトラフィックや処理時間の増加などを引き起こす．

本稿では，ノード間の通信データ量を削減することを目的とした，Reduce タスク割り当て手法について述べる．本手法は，以下の特徴を持つ．

- 各ノードが格納しているデータ量を考慮し，タスクの割り当てを行う．
- クラスタ全体で入力データのコピーに必要となる通信データ量を考慮し，タスクの割り当てを行う．

タスクの入力となるデータのコピーに必要な通信データ量を考慮するために，Map タスクのパーティション毎の出力データ量を格納し取得する．これらを考慮することで 1 つの Map タスクの出力が 1 つの Reduce タスクのみの入力となるときは，通信データ量が最少となるタスクの割り当てが行える．また，Map タスクの出力がノード間で偏っている時は，ノード間でのコピーが必要になるデータ量が最少となるようタスクを割り当てることで，通信データ量を削減でき，処理時間も短縮される．

以下本論文では，第 2 章で研究背景，3 章で提案する Reduce タスク割り当て手法，第 4 章で提案手法の Hadoop への実装について述べ，第 5 章で提案手法の評価を行う．

2. 研究背景

インターネット利用者の増加や，環境モニタリング，ライフログなどの普及により，蓄積データは増加の一途を辿っている．これらの大規模データを高速に処理するためには，複数台の計算機を協調動作させ，並列に処理することが有効である．並列分散処理には通信処理や同期処理などが必要になるが，これらを行うための機能を提供し，アプリケーション作成の負荷を軽減するためのシステム [1][2][3] が提案されている．Hadoop は，このような並列分散処理を容易に行うための環境を提供するシステムのひとつである．

2.1 Hadoop

Hadoop は，Apache Software Foundation によって開発されている，分散型コンピューティングのためのオープンソースソフトウェアである．Hadoop は大別して，高スループットでのアクセスを可能とする分散ファイルシステム Hadoop Distributed File System と，並列分散処理フレームワーク Hadoop MapReduce から構成される．

Hadoop MapReduce は，大規模データを処理するために Google が開発した並列分散処理フレームワーク MapReduce を基に実装したものである．MapReduce では，フレームワークによって，分散処理に必要な処理の割り当てや，ノード間の通信，障害への対応などの機能が提供されるため，ユーザーは容易に並列分散処理を実現できる．MapReduce は多くの企業で利用されており，また，データベース [7][8]，画像処理 [9]，計算科学アプリケーション [10]，機械学習 [11] など様々な分野での応用が研究されている．

MapReduce は，データ処理を独立に実行可能な複数のタスクに分割し，各ノードに分散させることで並列に処理を行う．タスクは，データの読み込み，整理を行い，中間データを出力する Map タスクと，その中間データを集約し最終的な結果を出力する Reduce タスクに分割される．中間データは各 Reduce タスクに対応するパーティションというグループに分割される．Reduce タスクは入力となるパーティションをコピーし，それを処理する．タスクは 1 台のホストノードによって，スレーブノードに割り当てられる．スレーブノードは，割り当てられたタスクに必要な入力データを，他のノードからコピーをする．このときのノード間で送受信されるデータ量が大きくなれば，処理時間も増加するため，通信データ量を小さく抑えるためのタスクスケジューリングが必要となる．

2.2 タスクスケジューリング

既存の Hadoop における Map タスクのスケジューリングでは，データのコピーにかかる時間を抑えるため，入力データのネットワーク位置を考慮したスケジューリングが行われる．タスクに必要な入力データがタスクを処理するノードに存在している場合をノードローカル，データが同一ラック内の他のノードに存在している場合をラックローカル，それ以外のノードに存在している場合をノンローカルと呼ぶ．ノードローカルでない場合，他のノードからネットワークを介したデータのコピーが必要となる．Map タスクのスケジューリングでは，ノードにタスクを割り当てる際に，ノードローカル，ラックローカル，ノンローカルの順にタスクを探索し，割り当てを行う．これにより，入力データがノードに近いタスクから優先的に割り当てられ，データのコピーにかかる時間を抑えることができる．また，Map タスクのスケジューリングをより効率よく行うための研究として，スケジューラが配置するタスクをノードの Map タスク数や入力データのレプリケーション構造に基づいて選択する手法 [12]，ノードローカルなタスクを配置できない場合にノンローカルなタスクを配置するかを判断する手法 [13]，CPU の I/O 待ち率が高い場合に動的に Map スロット数を増加させるスロットスケジューリング手法 [14][15] などが提案されている．

一方，Hadoop における Reduce タスクのスケジューリングでは，Map タスクのような入力データの位置を考慮したスケジューリングは行われておらず，タスクを要求したノードに無条件で割り当てを行う．Reduce タスクは Map タスクによって出力されたデータを入力とする．処理の内容や処理するデータによっては，ノード間で Map タスクが出力するデータに偏りが生じ，割り当てる Reduce タスクが適切でないとネットワークトラフィックが増加し，処理時間が長くなる場合がある．

このような問題に対処するために，ネットワーク距離とパーティションデータ量から，Reduce タスクの最も通信コストの小さくなる重心ノードを求め，割り当てを行う手法 CoGRS[16] が提案されている．CoGRS では，ノードがタスクを要求した際に，未実行の全ての Reduce タスクの重心ノードを求める．重心ノードはパーティションをコピーするノード間のネットワーク距離とコピーするデータ量から算出される．そして，タスクを要求したノードが重心ノードとなる Reduce タスクを割り当てる．タスクを要求したノードが重心ノードとなる Reduce タスクが存在しない場合，重心ノードとなる全てのノードの進捗状況を確認し，すぐにタスクを実行できないノードが重心ノードとなる Reduce タスクの中から割り当てを行う．このとき，タスクを要求したノードと重心ノードとのネットワーク距離が最も近いものが選択される．

提案手法では，Reduce タスクのスケジューリングにおいて，各ノードが格納するパーティションデータ量から，タスクを割り当てた際に必要となる通信データ量を算出し，クラスタ全体で最も通信データ量が少なくなるように Reduce タスクの割り当てを行う．

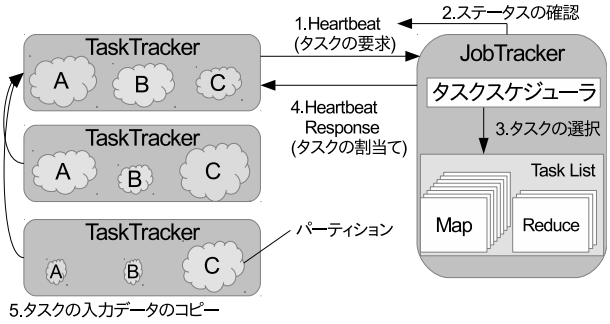

図 1　Hadoop におけるタスクスケジューリング

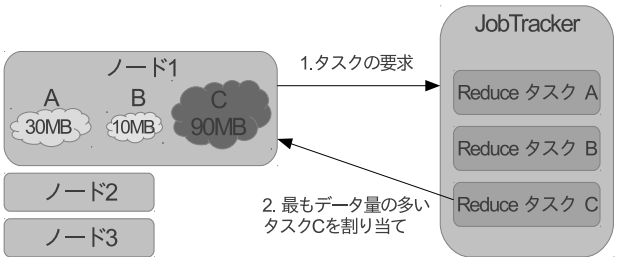

図 2　最大格納データ割り当て概略図

3. 通信データ量を考慮した Reduce タスク割り当て手法

Hadoop におけるタスクスケジューリングの流れを図 1 に示す．Hadoop でのタスクの割り当ては，ホストノードとなる JobTracker とスレーブノードとなる TaskTracker との Heartbeat 通信を利用して行われる．TaskTracker はタスクの実行が可能であることを Heartbeat 通信を用いて JobTracker に通知する．JobTracker は TaskTracker のステータスを確認すると，タスクリストの中からタスクを選択し，HeartbeatResponse によって割り当てを行う．タスクの入力となるデータが他のノードにある場合，ネットワークを介したデータのコピーが必要となる．Reduce タスクの入力データは，Map タスクで出力された中間データを各 Reduce タスクに対応するパーティションというグループに分けられたものである．各パーティションのデータサイズが均一である保証はなく，入力データや処理によって偏りが生じる場合がある．

既存の Hadoop における Reduce タスクの割り当ては，タスクを要求する TaskTracker に未実行のタスクのリストの先頭からタスクを 1 つ取り出し，無条件に割り当てるというものである．この手法では，ノード間での多くのデータの送受信が必要となるタスクの割り当てが行われ，効率が低下する可能性がある．本章では，Reduce タスクが入力に必要とするデータのコピーにかかる通信データ量を削減するための 2 つの手法について述べる．

3.1 最大格納データ量割り当て

最大格納データ量割り当ての概要を図 2 に示す．最大格納データ量割り当てでは，タスクを要求したノードが格納しているパーティションのデータ量を考慮し，最もデータ量の多いパーティションを入力とする Reduce タスクを割り当てる．図 2 の例だと，Reduce タスク A，B，C があり，ノード 1 はパーティション A のデータを 30MB，B のデータ 10MB，C のデータを 90MB 格納しており，最もデータ量の多い C のタスクが割り当てられる．これにより，ノードがデータをあまり格納していないパーティションを入力とする Reduce タスクが割り当てられることを防ぐことができるため，通信データ量の削減を期待できる．また，1 つの Map タスクの出力が 1 つの Reduce タスクのみの入力となるような処理では，Map タスクを実行したノードが Reduce タスクに必要なパーティションを全て格納するため，ノードローカルなタスクの割り当てが行える．

3.2 最少通信データ量割り当て

最大格納データ量割り当てでは，タスクを割り当てる際に他のノードのパーティションのデータ量を考慮していないため，タスク要求したノードが格納しているパーティションのデータ量が，他のノードと比べて少ない場合でも Reduce タスクを割り当ててしまい，通信データ量が多くなる可能性がある．この問題を解決するため，クラスタ全体で必要となる通信データ量が最少となるよう割り当てを行う最少通信データ量割り当てを提案する．

最小通信データ量割り当ての概要を図 3 に示す．最少通信データ量割り当てでは，全ての Reduce タスクをノードにどのように割り当てるかを考える．タスクをノードに割り当てた場合のデータのコピーに必要となる通信データ量を算出し，全てのノードとタスクの割り当ての組み合わせの中で，最も通信データ量が少ない組み合わせを求め，その結果に基づいてタスクの割り当てを行う．図 3 の例では，ノードが 3 台あり，それぞれがパーティション A，B，C のデータを格納している．各ノードに 1 つの Reduce タスクが割り当てられるものとし，通信データ量が最小となるタスクの割り当ての組み合わせを考える．ノード 1 に Reduce タスク A を割り当てた場合の通信データ量は，ノード 2，ノード 3 が格納するパーティション A のデータ量の合計である．タスク A，B，C の割り当ての組み合わせで必要となる通信データ量の合計を算出し，最少となる組み合わせを求める．その結果，ノード 1 にタスク B，ノード 2 にタスク A，ノード 3 にタスク C を割り当てることが最適な組み合わせとなり，タスクを要求したノードに対応するタスクを割り当てる．

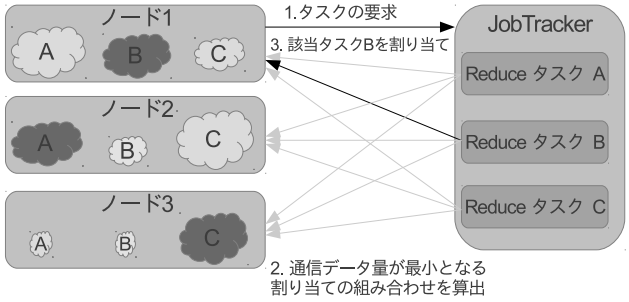

図 3　最小通信データ量割り当て概略図

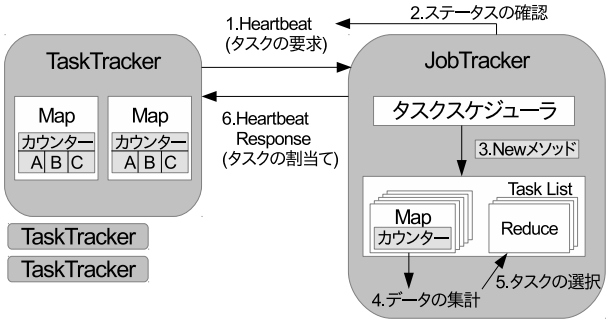

図 4　提案手法におけるタスクの割り当て

4. 実装

提案手法を，Hadoop-1.2.1 に実装した．提案手法のタスク割り当ての流れを図 4 に示す．各ノードのパーティション毎のデータ量を取得するため，Map タスクにカウンターを追加した．また，タスクスケジューラの Reduce タスクスケジューリング機能部分に変更を加え，各割り当て手法を実装した．提案手法のタスク割り当ての流れと，各手法の実装について述べる．

4.1 Reduce タスク割り当ての流れ

JobTracker から TaskTracker への Reduce タスクの割り当ての流れは以下のようになる．

(1) TaskTracker が Heartbeat 通信を用いて JobTracker にタスクを要求する
(2) JobTracker は，TaskTracker のステータスを確認し，タスクスケジューラを呼び出す
(3) タスクスケジューラは Reduce タスクを割り当てるかどうかを判断し，新たに実装した各手法のメソッドを呼び出す
(4) 全 Map タスクの進捗情報から，ノードが格納するパーティション毎のデータ量を集計する
(5) 集計したデータから，各手法を用いて Reduce タスクを選択し，JobTracker へ返す
(6) JobTracker はタスクを HeartbeatResponse に格納し，TaskTracker へ割り当てる

Hadoop MapReduce では，TaskTracker は RPC で JobTracker の Heartbeat メソッドを呼び出す．TaskTracker はタスクが実行可能かどうかを acceptNewTasks というフラグを用いて通知する．JobTracker はこのフラグを確認し，実行可能ならタスクスケジューラを呼び出す．タスクスケジューラは，実行中のジョブを表すクラス JobInProgress のメソッドを呼び出し，実行するべきタスクがあるか，Map タスクが一定数 (デフォルトで 5%) 終了しているかを確認した後，findTaskFromList メソッドの変わりに，新たに実装した各手法のメソッドを呼び出す．各メソッドは，全 Map タスクの進捗情報から，ノードが格納するパーティション毎のデータ量を集計し，各手法を用いて，割り当てるタスクを選択し，JobTracker に返す．JobTracker は返された Reduce タスクを HeartbeatResponse に格納し，TaskTracker に渡すことでタスクの割り当てを行う．

ノードが格納するパーティション毎のデータ量を取得するため，Map タスクに，中間データをファイルに出力する際にパーティション毎のデータ量を保持するカウンターを追加した．カウンターはタスクの進捗情報として，Map タスクから TaskTracker に渡され，Heartbeat 通信により JobTracker に渡される．タスクの進捗情報には，どのノードが実行しているか，処理は完了しているか等の情報が含まれており，これにより JobTracker は，各 Map タスクを実行したノードと出力されたパーティション毎のデータ量を知ることができる．

4.2 最大格納データ量割り当ての実装

最大格納データ量割り当てでは，まず全ての Map タスクの進捗情報から，タスクを要求したノードが格納しているパーティション毎のデータ量を集計する．各パーティションのデータ量を比較していき，未実行の Reduce タスクのリストの中で，タスク要求先のノードが最もデータを多く格納しているパーティションを入力とする Reduce タスクを求める．そして JobTracker に，当該 Reduce タスクを返す．必要な計算は最大値を求めるだけであり，計算量は未実行の Reduce タスクの数に依存する．処理が進むにつれて Reduce タスクは各ノードへ割り当てられていくため，未実行の Reduce タスクは減っていき，計算量も減少していく．通常，Reduce タスクの数は TaskTracker の数以下に設定されるため，計算量は少ない．

4.3 最少通信データ量割り当ての実装

最少通信データ量割り当てでは，ジョブの初期化時に，全てのノードとタスクの組み合わせ方の情報を持つ組み合わせリストを作成しておく．タスクの選択では，まず，全ての Map タスクの進捗情報から，全てのノードが格納

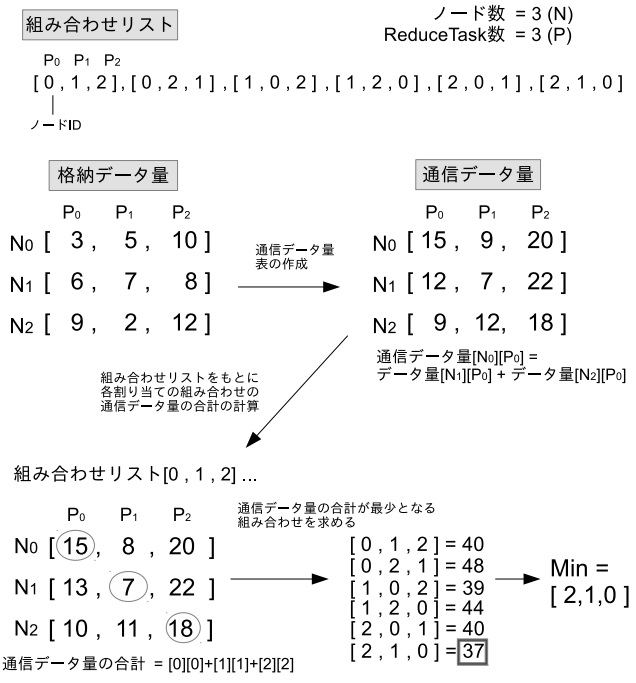

図 5　最少通信データ量割り当ての計算アルゴリズム

しているパーティション毎のデータ量を集計する．集計したデータから，ノードに Reduce タスクを割り当てた際に必要となる通信データ量の表を作成する．次に，初期化時に作成しておいた組み合わせリストをもとに，各組み合わせで必要となる通信データ量を計算する．全ての組み合わせを調べ，通信データ量が最少となるノードとタスクの組み合わせを求める．その結果に基づき，タスクを要求したノードに割り当てるタスクを JobTracker へ返す．

ノード数 3，Reduce タスク数 3 の場合の計算アルゴリズムを図 5 に示す．まず，全てのノードとタスクの割り当ての組み合わせを網羅するため，${}_3P_3$ の順列の要素を求め，組み合わせリストを生成する．タスクの選択では，全てのノードが格納しているパーティション毎のデータ量を集計し，格納データ量表を作成する．格納データ量表から，ノードにタスクを割り当てたときにデータのコピーに必要となる通信データ量表を作成する．例えば，ノード N_0 にタスク P_0 を割り当てたときに必要となる通信データ量は，その他のノード N_1，N_2 が格納している P_0 のデータ量の合計となる．次に，初期化時に作成しておいた組み合わせリストをもとに，各組み合わせで必要となる通信データ量を計算する．組み合わせが [0,1,2] の場合は，タスク P_0 をノード N_0 に，タスク P_1 をノード N_1 に，タスク P_2 をノード N_2 に割り当てたときに必要となる通信データ量である．全ての組み合わせを調べ，組み合わせ [2,1,0] が通信データ量が最少となる割り当てとなるため，これを選択する．

通信データ量が最少となるノードとタスクの組み合わせの計算は，TaskTracker がタスクを要求する度に行う．

表 1　評価環境

Hadoop	hadoop-1.2.1
OS	Debian 3.11.10-1
CPU	Intel(R) Xeon(R) CPU E31260L @ 2.40GHz
メモリ	8GB
ノード数	10
ネットワーク	1Gbps Ethernet

表 2　入力データとタスク数

プログラム	入力データサイズ	Map タスク数	Reduce タスク数
terasort	5.3GB	90	7
wordcount	5.3GB	90	7
pagerank(stage1)	600MB(300 万ページ)	90	7
pagerank(stage2)	(stage1 の出力)	7	7

Reduce タスクの割り当ては，通常全 Map タスクの 5%が終了した段階から開始される．これは，Reduce タスクがそれぞれの Map タスクが処理を終えたらすぐにデータのコピーを開始するためである．そのため，Map タスクの処理が進むにつれて各ノードの格納データ量が増加していき，通信データ量が最少となる組み合わせが変わる場合がある．よって，TaskTracker がタスクを要求する度に，その時点でタスクを実行可能なノードと未実行の Reduce タスクの中で，最も通信データ量の少なくなる組み合わせを計算し，最適なタスクを割り当てる．計算量は，ノード数を N，Reduce タスク数を R としたときの組み合わせの数 ${}_NP_R$ に依存するため，ノードとタスクの数によっては非常に大きくなる．また，組み合わせを網羅するためのリストを作成する際にも多くの時間がかかる．

5. 評価

Hadoop に付属している terasort ，wordcount プログラムと Intel の提供する Hadoop のベンチマークである HiBench[17] の中の PageRank プログラムを使用し，各手法での通信データ量について評価を行った．評価環境を表 1 に示す．JobTracker ノード 1 台と TaskTracker ノード 9 台の計 10 台のクラスタ上で評価を行った．terasort プログラムは，1 つの MAP タスクは 1 つのパーティションにしか出力せず，各ノードの格納するパーティション毎のデータ量に偏りが生じる．wordcount プログラムは，1 つの MAP タスクがほぼ全てのパーティションへ出力し，ノード間でのデータの偏りがほとんど起こらない．PageRank プログラムは，Web ページのランク付けをするプログラムであり，Web ページのリンク関係と現在のランクを算出する stage1 と，ページ毎にリンク先のランクとリンク数からランクを計算する stage2 の 2 つの処理を行う．stage1 の Map タスクは全てのパーティションへデータを出力し，ノード間の格納データの偏りはほとんど起こらない．stage1 の Reduce タスクの出力が stege2 の Map タスクの

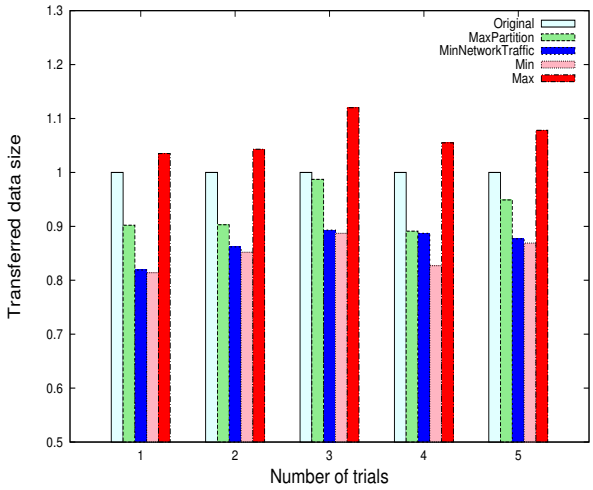

図 6　terasort での通信データ量

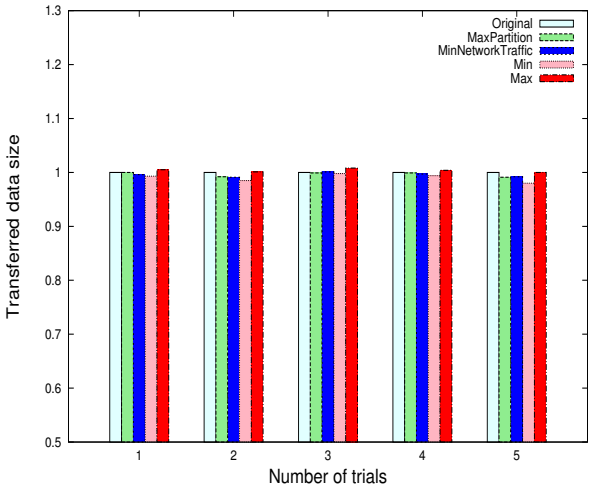

図 7　wordcount での通信データ量

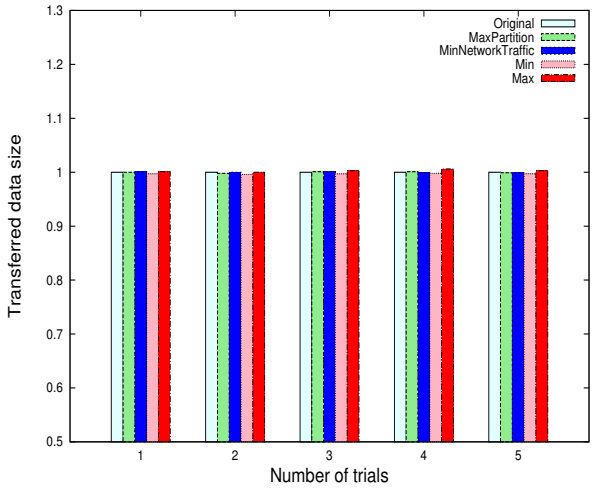

図 8　pagerank stage1 での通信データ量

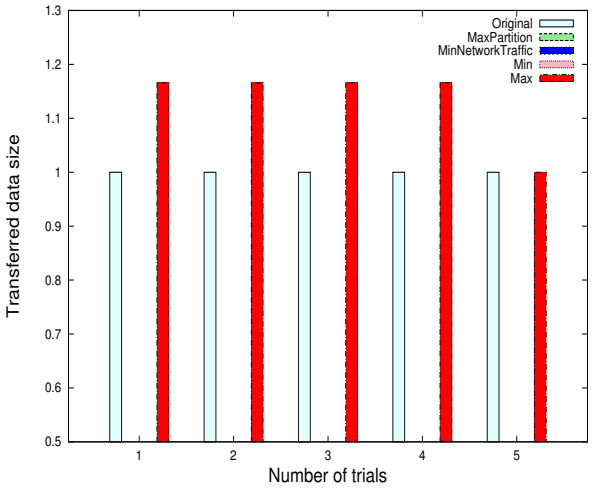

図 9　pagerank stage2 での通信データ量

入力となる．stage2 では，Map タスクと Reduce タスクの数が等しく，1 つの Map タスクの出力が 1 つの Reduce タスクの入力となる．

本評価での入力データとタスク数を表 2 に示す．terasort, wordcount プログラムの入力には約 5.3GB のログデータを用いた．PageRank プログラムの入力には，HiBench の付属ツールで生成したページ数 300 万のデータセット使用した．データ量とノードへの分散を考慮して，Map タスク数は TaskTracker 数の 10 倍の 90 とした．通常，Reduce タスクの数は TaskTracker の数以下に設定されるが，本実験では 7 とした．

5.1　通信データ量の評価結果

各評価プログラムを実行したときの通信データ量を測定した．測定結果を図 6-図 9 に示す．Original は Hadoop のタスク割り当てを，MaxPartition は最大格納データ量割り当てを，MinNetworkTraffic は最少通信データ量割り当てを表す．全ての Map タスクの出力が完了した後の，最終的なパーティションのデータ量から，最少と最大の通信データ量を求めたものを，それぞれ Min，Max とした．縦軸は Original の通信データ量を 1 としたとき通信データ量，横軸は試行回数 5 回のそれぞれの結果となっている．表 3 に平均通信データ量を示す．terasort プログラムでは Min と Max で通信データ量に約 1.13GB の差がある．Original と比べて，最大データ格納割り当てでは平均 8%，最大 11%，最少通信データ量割り当てでは，平均 13%，最大 18%の通信データ量が削減している．wordcount および pagerank stage1 では，通信データ量の最小値と最大値の差がほとんどなく，各手法において通信データ量は同じとなった．pagerank の stage2 では，2 つの提案手法ともに通信データ量は 0 となり 2.79 GB 削減している．terasort，pagerank の stage2 では，各ノードが格納するパーティション毎のデータ量に偏りが起こり，割り当てる Reduce タスクによって通信データ量が大きく変動するため，各手法が有効に働いたと考えられる．特に pagerank stage2 のような 1 つの Map タスクの出力が 1 つの Reduce タスクのみの入力になる場合には，通信データ量を 100%削減でき，効果が大きい．一方，wordcount，pagerank の stage1

表 3　平均通信データ量 (GB)

プログラム	Original	MaxPartition	MinNetworkTraffic	Min	Max
terasort	5.34	4.91	4.64	4.50	5.63
wordcount	8.85	8.82	8.81	8.76	8.89
pagerank(stage1)	1.30	1.30	1.30	1.28	1.30
pagerank(stage2)	2.79	0.00	0.00	0.00	3.16

表 4　terasort の平均処理時間 (秒)

Reduce タスク数	Original	MaxPartition	MinNetworkTraffic
4	61.317	59.166	58.165
5	56.188	56.160	55.177
6	47.150	46.143	45.000
7	47.153	45.143	44.037
8	42.147	41.226	41.043
9	41.140	39.141	48.905
10	44.148	42.149	60.871

では，各ノードが格納するパーティション毎のデータ量に偏りがほとんどなく，どのノードにどのタスクを割り当てても通信データ量はほとんど変わらないため，両手法ともに効果は少なかった．

5.2　処理時間の評価結果

TaskTracker 数を 10 台とし，Reduce タスクの数を 4 から 10 に変化させて terasort プログラムを実行し，各手法での処理時間を測定した．測定結果を表 4 および図 10 に示す．グラフの縦軸は Original の処理時間を 1 としたときの割合，横軸は，Reduce タスクの数となっている．MaxPartition は全ての場合で Original より処理時間が削減している．MinNetworkTraffic においても Reduce タスクの数が 4 から 8 の時は処理時間が削減しており，3 つの手法の中で最も処理時間が短くなっている．このことから，通信データ量が削減されたことで，パーティションのコピーにかかる時間が短縮され，処理時間の削減に繋がっていることが分かる．そして，クラスタ全体の通信データを考慮したことで，削減率の高まった MinNetworkTraffic が最も処理時間が短くなった．しかし，MinNetworkTraffic では，Reduce タスクの数が 9 以上になると処理時間が大きく増加している．このときの，組み合わせの候補の数は ${}_{10}P_9$=3628800 となり，候補の生成時間および最小値の計算時間が長くなったことが原因だと考えられる．よって，組み合わせ候補数が少ない場合は最少通信データ量割り当てを使用し，候補数が多いときは最大格納データ量割り当て，もしくは別の最少通信データ量の近似解を求めるような手法を使用するのが良いと考えられる．

6.　おわりに

本論文では，MapReduce における通信データ量を考慮した Reduce タスク割り当て手法について述べた．最大格納データ量割り当てでは，タスクを要求したノードが格納するパーティションのデータ量を考慮し，最もデータ量の多いパーティションに対応する Reduce タスクを割り当てる．最少通信データ量割り当てでは，クラスタ全体でパーティションのコピーに必要となる通信データ量を考慮し，通信データ量が最少となるノードと Reduce タスクの組み合わせを求めて，Reduce タスクを割り当てる．両手法ともに，各ノードが格納するパーティション毎のデータ量に偏りがある場合には，通信データ量および処理時間を削減できる．最大通信データ量割り当ては，ノード数，タスク数に関わらず通信データ量および処理時間を削減できるが，最少通信データ量割り当てと比べて削減量が小さい．最少通信データ量では，組み合わせの候補が少ないときは，通信データ量および処理時間を大きく削減できる．しかし，組み合わせの候補数が増加するにつれて計算量が大きくなり，組み合わせ候補と最小値の計算時間が増加してしまう．組み合わせ候補のリストを格納するためのメモリ領域も大きくなる．また，今回は 1 つのラックのみの実験環境であった．実際の環境ではノード数がもっと多く，複数のラックで分けられていることもあるため，ネットワークの距離等の影響も考慮する必要がある．

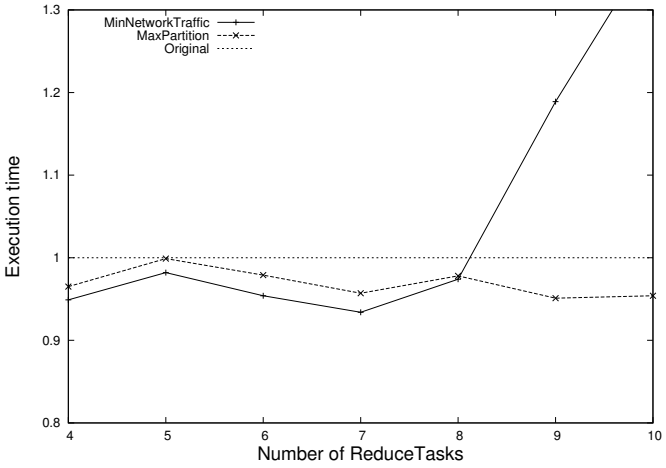

図 10　terasort の処理時間

参考文献

[1] OpenMPI. http://www.mcs.anl.gov/research/projects/mpi
[2] 原 健太朗, 田浦 健次朗, 近山 隆. DMI：計算資源の動的な参加/脱退をサポートする大規模分散共有メモリインターフェース．情報処理学科論文誌，Vol. 3, No. 1, pp. 1-40 (2010).
[3] 三添匠, 小鍛治翔太, 芝公仁. プロセスを分散実行するためのシステムコール制御手法, 情報処理学会研究報告. [ハイパフォーマンスコンピューティング] 2011- HPC-132(33), 1-7, 2011-11-21
[4] Hadoop. http://hadoop.apache.org/
[5] Dean, J. and Ghemawat, S. 2004. MapReduce: Simplified data processing on large clusters. In Proceedings of Operating Systems Design and Implementation(OSDI). San Francisco, CA. 137-150.
[6] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. In 19th Symposium on Operating Systems Principles, pages 29-43, Lake George,

New York, 2003.
[7] Hbase. http://hbase.apache.org/
[8] Hive. http://hive.apache.org/
[9] 塚本 勢児, 布広 永示. Hadoop を用いた衛星画像データ解析処理の高速化の研究. 研究報告システムソフトウェアとオペレーティング・システム（OS）2014-OS-128, 9, 1-6, 2014-02-27
[10] 滝澤 真一朗, 松田 元彦, 丸山 直也. MapReduce による計算科学アプリケーションのワークフロー実行支援. ハイパフォーマンスコンピューティングと計算科学シンポジウム論文集, 2014, 1-9 (2013-12-31)
[11] Mahout. http://mahout.apache.org/
[12] Ibrahim, S., Jin, H., Lu, L., He, B. Antoniu, G. and Wu,S. Maestro: Replica-Aware Map Scheduling for MapReduce, 12th IEEE/ACM Internatinal Symposium on Cluster, I Cloud and Grid Computing, Ottawa, Canada(2012).
[13] Zhang, X., Feng Y., Feng, S, Fan, J. and Ming, Z. An effective data locality aware task scheduling method for MapReduce framework in heterogeneous environments International Conference on Cloud and Service Computing (CSC), (2011)
[14] Kurazumi, S., Tsumura, T., Saito, S., Matsuo, H. Dynamic Processing Slots Scheduling for I/O Intensive Jobs of Hadoop MapReduce , ICNC '12 Proceedings of the 2012 Third International Conference on Networking and Computing , Pages 288-292 , DOI: 10.1109/ICNC.2012.53
[15] 藏澄汐里 , 津邑公暁 , 齋藤彰一 , 松尾啓志. Hadoop MapReduce におけるリソース使用状態を考慮したスロットスケジューリング. 研究報告ハイパフォーマンスコンピューティング（HPC）,2013-HPC-142(32),1-8
[16] M. Hammoud, M.S. Rehman, and M.F. Sakr. Center-of-gravity reduce task scheduling to lower mapreduce network traffic. In 2012 IEEE 5th International Conference on Cloud Computing (CLOUD) , pp. 49 58, June 2012.
[17] HiBench. https://github.com/intel-hadoop/HiBench
[18] Tom White: Hadoop 第 3 版 , オライリー・ジャパン , July 2013.